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

Ricardo M. A. Silva*, Mauricio G. C. Resende†, Panos M. Pardalos‡ and João L. Facó§
* Centro de Informática, Universidade Federal de Pernambuco, Recife, PE, Brazil.
email: rmas@cin.ufpe.br
† Algorithms and Optimization Research Department, AT&T Labs Research, Florham Park, NJ 07932 USA.
email: mgcr@research.att.com
‡ Department of Industrial and Systems Engineering, University of Florida, FL 32611, USA.
email: pardalos@ufl.edu
§ Departamento de Ciência da Computação, Universidade Federal do Rio de Janeiro, RJ, Brazil.
email: jldfaco@ufrj.br

***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 bound-constrained continuous global optimization problems subject to nonlinear constraints. Experimental results illustrate its effectiveness on some functions from CEC2006 benchmark (Liang et al. [2006]).**

I. Introduction

Continuous 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 $f : S \rightarrow R$. In this paper, we consider the domain S as the intersection between a set of nonlinear constraints and a hyper-rectangle $\Omega = \{x = (x_1, \ldots, x_n) \in R^n : \ell \leq x \leq u\}$, where $\ell \in R^n$ and $u \in R^n$ such that $u_i \geq l_i$, for $i = 1, \ldots, n$, in order to present the BRKGA heuristic for solving bound-constrained continuous global optimization problems subject to nonlinear constraints:

$$minf(x),\;\; x = (x_1, x_2, \ldots, x_n) \tag{1}$$

subject to:

$$g_i(x) \leq 0,\;\; i = 1, \ldots, q \tag{2}$$

$$h_j(x) = 0,\;\; j = q+1, \ldots, m \tag{3}$$

$$l_i \leq x_i \leq u_i,\;\; i = 1, \ldots, n \tag{4}$$

Given that the constraints (2) can be written as equalities with the introduction of the q slack variables $x_{n+1}, \ldots, x_{n+q}$:

$$g_i(x_1, \ldots, x_n) + x_{n+i} = 0,\;\; i = 1, \ldots, q, \tag{5}$$

with $x_k \geq 0$, $k = n+1, \ldots, n+q$, the original problem can be reduced to the minimization of function $F(x_1, \ldots, x_{n+q})$ defined as follows:

$$[f(x_1, \ldots, x_n) - f^*]^2 + \sum_{i=1}^{q} [g_i(x_1, \ldots, x_n) + x_{n+i}]^2 + \sum_{j=q+1}^{m} [h_j(x_1, \ldots, x_n)]^2$$

(6)

subject to:

$$l_i \leq x_i \leq u_i,\;\; i = 1, \ldots, n \tag{7}$$

$$x_k \geq 0,\;\; k = n+1, \ldots, n+q, \tag{8}$$

where f^* is a known optimum value of problem (1-4), or the best known value in the literature. In case we do not know what the global solution value is, if possible, we can consider an appropriate lower bound as f^*.

Since $F(x_1, \ldots, x_{n+q}) \geq 0$ for all $l_i \leq x_i \leq u_i$, $i = 1, \ldots, n$, and $x_k \geq 0, k = n+1, \ldots, n+q$, it is easy to see that $F(x_1, \ldots, x_{n+q}) = 0 \iff f(x_1, \ldots, x_n) = f^*$; $g_i(x_1, \ldots, x_n) = x_{n+i}$, $i = 1, \ldots, q$; and $h_j(x_1, \ldots, x_n) = 0$, $j = q+1, \ldots, m$. Hence, we have the following: $\exists\; z^* = (x_1, \ldots, x_{n+q})^*$ feasible $\ni F(z^*) = 0 \implies z^*$ is a global minimizer of problem (1-4).

This paper is organized as follows. BRKGA heuristic for global optimization problem is described in Sections II and III. In Section IV, experimental results illustrate the effectiveness of BRKGA for global optimization with nonlinear constraints. Concluding remarks are made in Section V.

II. Biased random-key genetic algorithms

Genetic algorithms with random keys, or *random-key genetic algorithms* (RKGA), were first introduced by Bean [1994] 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.

Figure 1 illustrates the evolution dynamics. On the left of the figure is the current population. After all individuals are sorted by their fitness values, the best fit are placed in the elite partition labeled ELITE and the remaining individuals are placed in the partition labeled NON-ELITE. The elite random-key vectors are copied without change to the partition labeled TOP in the next population (on the right side of the figure). A number of mutant individuals are randomly generated and placed in the new population in the partition labeled BOT. The remainder of the population of the next generation is completed by crossover. In a RKGA, Bean [1994] selects two parents at random from the entire population. A *biased random-key genetic algorithm,* or BRKGA [Gonçalves and Almeida, 2002, Ericsson et al., 2002, Gonçalves and Resende, 2004], differs from a RKGA in the way parents are selected for mating. In a BRKGA, each element is generated combining one element selected at random from the partition labeled ELITE in the current population and one from the partition labeled NON-ELITE. 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 the *parameterized uniform crossover* [Spears and DeJong, 1991], 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 $1 - \rho_e$. Figure 2 illustrates the crossover process for two random-key vectors with four components each. Chromosome 1 refers to the elite individual and Chromosome 2 to the non-elite one. In this example the value of $\rho_e = 0.7$, i.e. the offspring inherits the component of the elite parent with probability 0.7 and of the other parent with probability 0.3. A randomly generated real in the interval $[0, 1]$ simulates the toss of a biased coin. If the outcome is less than or equal to 0.7, then the child inherits the component of the elite parent. Otherwise, it inherits the component of the other parent. 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.

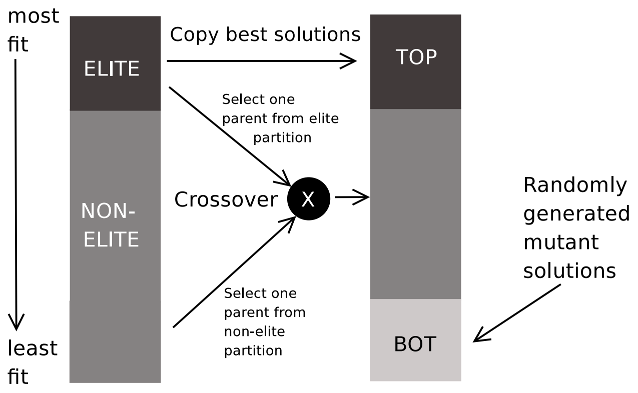
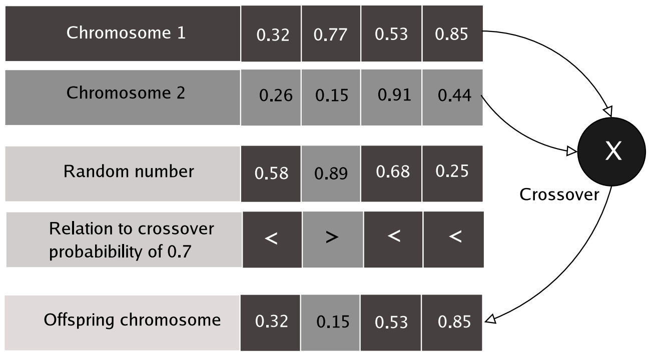

most fit
least fit
ELITE
NON-ELITE
Copy best solutions
Select one parent from elite partition
Crossover
X
Select one parent from non-elite partition
TOP
BOT
Randomly generated mutant solutions

Fig. 1. Transition from generation k to generation $k+1$ in a BRKGA.

BRKGA heuristics are based on a general-purpose meta-heuristic framework. In this framework, depicted in Figure 3, there is a clear divide between the *problem-independent* portion of the algorithm and the *problem-dependent* part. The problem-independent portion has no knowledge of the problem being solved. The only connection to the optimization problem being solved is the problem-dependent portion of the algorithm, where the decoder produces solutions from the vectors of random-keys and computes the fitness of these solutions. Therefore, to specify a BRKGA heuristic one need only define its chromosome representation and the decoder. To describe a BRKGA for nonlinearly-constrained global optimization problems, 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 Ω.
- *Decoding a solution from a vector of random keys*. A decoder takes as input the vector of random keys χ and returns a solution $x \in \Omega$ with $x_i = l_i + \chi_i \cdot (u_i - l_i)$, for $i = 1, \ldots, n$. After obtaining the solution $x \in \Omega$, we proceed by trying to improve it using the local search described in the next section. The solutions



Chromosome 1	0.32	0.77	0.53	0.85
Chromosome 2	0.26	0.15	0.91	0.44
Random number	0.58	0.89	0.68	0.25
Relation to crossover probability of 0.7	<	>	<	<
Offspring chromosome	0.32	0.15	0.53	0.85

Fig. 2. parameterized uniform crossover: mating in BRKGAs.

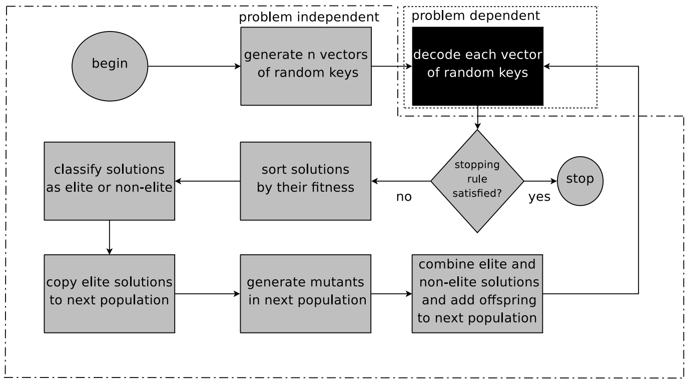

problem independent
problem dependent
begin
generate n vectors of random keys
decode each vector of random keys
classify solutions as elite or non-elite
sort solutions by their fitness
stopping rule satisfied?
no
yes
stop
copy elite solutions to next population
generate mutants in next population
combine elite and non-elite solutions and add offspring to next population

Fig. 3. Flowchart of a BRKGA

produced by the local search usually disagree with the genes initially supplied in the vector of random keys to the decoder. In these cases, in order to reflect the changes made by the local search phase of the decoder, the heuristic replaces the initial chromosome with the returned by the local search procedure, where $\chi_i = (x_i - l_i)/(u_i - l_i)$, for $i = 1, \ldots, n$. During all decoder process, the solutions fitness are calculated by the objective function $f : S \to \mathbb{R}$ of the global optimization problem.

III. Local improvement procedure

BRKGA makes no use of derivatives. Though derivatives can be easily computed for many functions, they are not always available or efficiently computable for all functions. The local improvement phase (with pseudo-code shown in Figure 4) can be seen as *approximating* the role of the gradient of the objective function $f(\cdot)$. From a given input point $x \in \mathbb{R}^n$, the local improvement algorithm generates a neighborhood, and determines at which points in the neighborhood, if any, the objective function improves. If an improving point is found, it is made the current point and the local search continues from the new solution.

Let $\bar{x} \in \mathbb{R}^n$ be the current solution and h be the current grid discretization parameter. Define $S_h(\bar{x}) = \{x \in \Omega \mid \ell \leq x \leq u,\ x = \bar{x} + \tau \cdot h,\ \tau \in Z^n\}$ to be the set of points in Ω that are integer steps (of size h) away from $\bar{x}$. Let $B_h(\bar{x}) = \{x \in \Omega \mid x = \bar{x} + h \cdot (x' - \bar{x})/\|x' - \bar{x}\|,\ x' \in S_h(\bar{x}) \setminus \{\bar{x}\}\}$ be the projection of the points in $S_h(\bar{x}) \setminus \{\bar{x}\}$ onto the hyper-sphere centered at $\bar{x}$ of radius h. The *h-neighborhood* of the point $\bar{x}$ is defined as the set of points in $B_h(\bar{x})$. The procedure takes as input a starting solution $x \in \Omega \subseteq \mathbb{R}^n$, the objective function $f(\cdot)$, lower and upper bound vectors ℓ and u, as well as the parameters h_s and h_e, the starting and ending grid discretization densities, respectively. The maximum number of points $\texttt{MaxPointsToExamine} \leq \prod_{i=1}^{n} \lceil (u_i - \ell_i)/h \rceil$ in $B_h(x^*)$ that are to be examined is also taken as an input parameter. If all of these points are examined and no improving point is found, the current solution x^* is considered an *h-local minimum*.

The current best local improvement solution x^* is initialized to x in line 1. In line 2, the objective function value f^* of the best solution found is initialized to $f(x)$. Next, the parameter h, that controls the discretization density of

```
procedure LocalImprovement($x, f(\cdot), h_s, h_e, \ell, u$, MaxPointsToExamine)
1     $x^* \leftarrow x$;
2     $f^* \leftarrow f(x)$;
3     $h \leftarrow h_s$;
4     Impr ← false;
5     while $h \geq h_e$ do
6         NumPointsExamined ← 0;
7         while NumPointsExamined ≤ MaxPointsToExamine do
8             $x \leftarrow$ RandomlySelectElement($B_h(x^*)$);
9             if $\ell \leq x \leq u$ and $f(x) < f^*$ then
10                $x^* \leftarrow x$;
11                $f^* \leftarrow f(x)$;
12                NumPointsExamined ← 0;
13                Impr ← true;
14            end if
15            NumPointsExamined ← NumPointsExamined + 1;
16        end while
17        if Impr = true then
18            return $x^*$;
19        else
20            $h \leftarrow h/2$;
21        end if
22    end while
23    return $x^*$;
end LocalImprovement;
```

Fig. 4. Pseudo-code for local improvement phase.

the search space, is initialized to h_s in line 3, and in line 4 the variable Impr is set to **false**. Starting at the point x^*, in the loop in lines 7–16, the algorithm randomly selects points in $B_h(x^*)$ (line 8), one at a time. In line 9, if the current point x selected from $B_h(x^*)$ is feasible and is better than x^*, then x^* is set to x (line 10), f^* is set to $f(x)$ (line 11), NumPointsExamined is set to zero (line 12), Impr is set to **true** (line 13), and the loop in lines 7–16 restarts with x^* as the starting solution. In line 17, if the variable Impr is still set to false, then in line 20 the grid density is increased by halving h, and the loop in lines 7–16 is re-initialized if $h \geq h_e$. Local improvement is terminated if an *h-local minimum* solution x^* is found. At that point, x^* is returned from the local improvement procedure in line 18 or 23.

IV. Experimental results

The experiments to follow were carried out 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. The implementation of BRKGA was done in the C++ programming language and compiled with gcc version 4.4.3. The algorithm used for random-number generation is an implementation of the Mersenne Twister algorithm described in Matsumoto and Nishimura [1998]. The boxplots were done in R [R Development Core Team, 2011]. For the experiments to follow, we made use of the test problems g01 [Floudas and Pardalos, 1990], g02 [Koziel and Michalewicz, 1999], g03 [Michalewicz et al., 1996], g04 [Himmelblau, 1972] and g05 [Hock and Schittkowski, 1981], due to their heterogeneous properties described in Table I.

In all five problems, we ran BRKGA 200 times (a different starting random number seed for each run from 270001 to 270200) with $p = 100$, $p_e = 0.2p$, $p_m = 0.1p$, $\rho_e = 0.7$, $h_s = 0.05$, $h_e = 0.00001$, $rho_{lo} = 0.15$, $\texttt{MaxPointsToExamine} = 1000$, and $\epsilon = 0.00001$. At any time during a run, we define the optimality gap by $GAP = F(x_1, \ldots, x_{n+q}) - F(z^*)$, where $(x_1, \ldots, x_{n+q})$ is the current best solution found by the heuristic and $F(z^*) = 0$. We then say that the heuristic

TABLE I. FOR THE FIVE PROBLEMS (g01-g05) FROM CEC2006 BENCHMARK [LIANG ET AL., 2006]: n IS THE NUMBER OF DECISION VARIABLES; $\rho = |F|/|S|$, THE ESTIMATED RATIO BETWEEN THE FEASIBLE REGION AND THE SEARCH SPACE; LI, NI, LE AND NE ARE THE NO. OF LINEAR INEQUALITY, NONLINEAR INEQUALITY, LINEAR EQUALITY AND NONLINEAR EQUALITY CONSTRAINTS, RESPECTIVELY; AND a, THE NO. OF ACTIVE CONSTRAINTS AT x.

Prob.	**n**	**type**	$f(x^*)$	ρ	**LI**	**NI**	**LE**	**NE**	**a**
g01	13	quadratic	-15.0000000000	0.0111	9	0	0	0	6
g02	20	nonlinear	-0.8036191042	99.9971	0	2	0	0	1
g03	10	polynomial	-1.0005001000	0.0000	0	0	0	1	1
g04	5	quadratic	-30665.5386717834	52.1230	0	6	0	0	2
g05	5	cubic	5126.4967140071	0.0000	2	0	0	3	3

has solved the problem if $GAP \leq \epsilon$ with $\epsilon = 0.00001$. Therefore, we consider a solution to be found if the objective function value becomes smaller than 10^{-5}. In each problem, the heuristic was able to find its optimal (or best known) solution in all 200 running.

Table II gives the minimum, 1st quartile, median, mean, 3rd quartile and maximum times (in seconds), as well as the standard deviation of the running times spent to find each optimal (or best know) solution for each problem. According to Table II, on average BRKGA found the solutions in less than 21.62, 120.20, 0.60, 134.20, 4.00 seconds for g01, g02, g03, g04, and g05 problems, respectively. In the worst (maximum) case, BRKGA found the solutions in less than 253.60, $1,076.00$, 0.88, 913.40, and 15.15 seconds, respectively.

We record the time taken to find the optimal (or best know) solution for each problem, in order to plot also its runtime distribution (or time-to-target plots [Aiex et al., 2002, 2006]) in Figure 5.

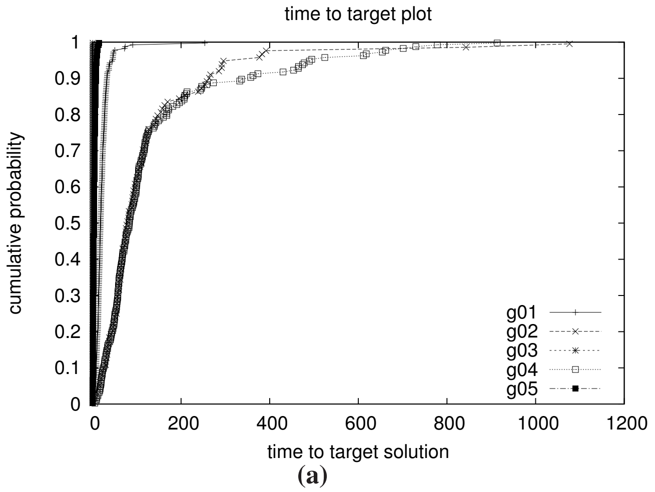
As can be seen in more details from Figure 5(b), a zoom of Figure 5(a), about 95% of the runs terminated in less than 46.00, 377.00, 0.77, 495.00, and 9.40 seconds for g01, g02, g03, g04, and g05 problems, respectively.

The differences in the distributions of running times for BRKGA to find the optimal (or best know) solution for each problem is shown by side-by-side boxplots in Figure 6.

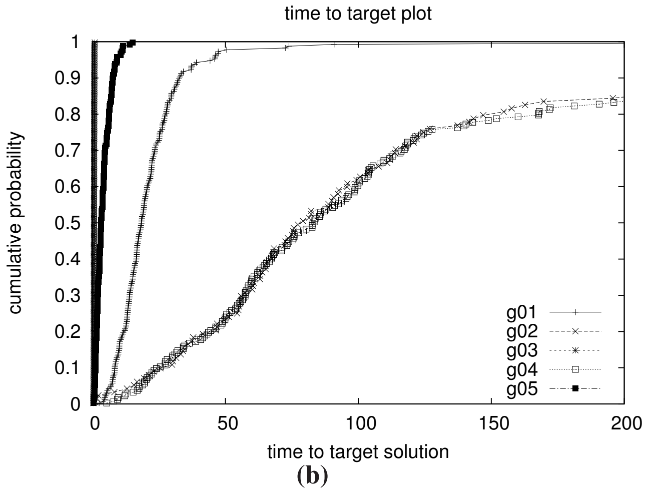
One can see at a glance from Figures 6(a) and (b) that, while 75% of running times (3rd quartile) spent by BRKGA to find the optimal (or best know) solutions for g01, g02, g03, g04, and g05 problems are smaller than 25.36, 123.90, 0.67, 125.00, and 5.60 seconds, respectively; 50% (median) of running times are less than 18.44, 78.24, 0.61, 83.25, and 3.28 seconds, respectively.

Finally, we compare the BRKGA heuristic with the continuous GRASP (C-GRASP) algorithm [Hirsch et al., 2007, 2010, Silva et al., 2012] on same suite of test problems g01–g05 [Facó et al., 2013]. The C-GRASP implementation used in this work is described in Silva et al. [2012]. In all five problems, we ran C-GRASP 200 times (a different starting random number seed for each run from 270001 to 270200). In each problem, the C-GRASP heuristic was able to find its optimal (or best known) solution in all 200 running.

Pseudo-code for C-GRASP is shown in Figure 7. The procedure takes as input the problem dimension n, lower and upper bound vectors ℓ and u, the objective function $f(\cdot)$, as well as the parameters h_s, h_e, and ρ_{lo}. Parameters h_s and h_e define the starting and ending grid discretization densities while parameter ρ_{lo} defines the portion of the neighborhood



(a)



(b)

Fig. 5. Plots of cumulative probability distributions (time-to-target plots) of BRKGA running times to find the optimal (or best known) solution of the g01-g05 problems.

of the current solution that is searched during the local improvement phase.

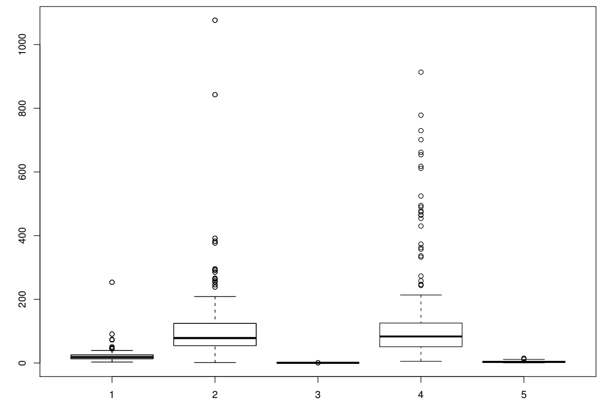
Line 1 of the pseudo-code initializes the objective function value f^* of the best solution found to infinity. Each time the stopping criteria of line 2 are not satisfied, another iteration takes place, as seen in lines 3–17. At each iteration, in line 3 the initial solution x is set to a random point distributed uniformly over the box in $\mathbb{R}^n$ defined by ℓ and u. The parameter h, that controls the discretization density of the search space, is re-initialized to h_s. The construction and local

TABLE II. BRKGA TIMES (IN SECONDS) TO TARGET SOLUTIONS OF g01-g05 PROBLEMS.

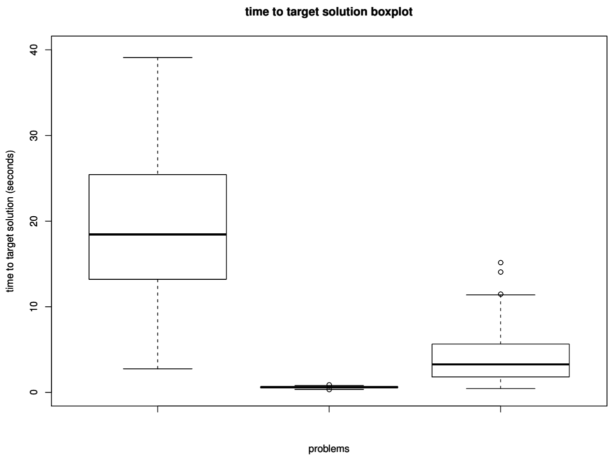
prob.	min	1st Qu.	median	mean	3rd Qu.	max	std. dev.
g01	2.76	13.24	18.44	21.62	25.36	253.60	20.324
g02	1.12	54.53	78.24	120.20	123.90	1076.00	145.316
g03	0.33	0.54	0.61	0.60	0.67	0.88	0.095
g04	5.37	51.38	83.25	134.20	125.00	913.40	157.995
g05	0.44	1.82	3.28	4.00	5.60	15.15	2.784

TABLE III. C-GRASP TIMES (IN SECONDS) TO TARGET SOLUTIONS OF g01-g05 PROBLEMS.

prob.	min	1st Qu.	median	mean	3rd Qu.	max	std. dev.
g01	8.72	9.91	11.22	12.08	13.20	19.32	2.793
g02	3.00	657.00	1055.00	1009.00	1321.00	1881.00	429.347
g03	0.02	0.06	0.09	0.10	0.13	0.27	0.060
g04	3.37	7.55	98.33	604.50	561.40	5472.00	1369.259
g05	1.88	2.49	3.46	3.43	4.45	5.04	1.057



(a)



(b)

Fig. 6. Boxplots of BRKGA running times to find the optimal (or best known) solution of the (a) g01-g05 problems, and (b) g01, g03, and g05 problems.

```
procedure C-GRASP($n, \ell, u, f(\cdot), h_s, h_e, \rho_{lo}$)
1   $f^* \leftarrow \infty$;
2   while Stopping criteria not met do
3       $x \leftarrow$ UnifRand($\ell, u$);
4       $h \leftarrow h_s$;
5       while $h \geq h_e$ do
6           $\text{Impr}_C \leftarrow$ false;
7           $\text{Impr}_L \leftarrow$ false;
8           $[x, \text{Impr}_C] \leftarrow$ ConstructGreedyRand.($x, f(\cdot), n, h, \ell, u, \text{Impr}_C$);
9           $[x, \text{Impr}_L] \leftarrow$ LocalImprovement($x, f(\cdot), n, h, \ell, u, \rho_{lo}, \text{Impr}_L$);
10          if $f(x) < f^*$ then
11              $x^* \leftarrow x$;
12              $f^* \leftarrow f(x)$;
13          end if
14          if $\text{Impr}_C$ = false and $\text{Impr}_L$ = false then
15              $h \leftarrow h/2$;    /* make grid more dense */
16          end if
17      end while
18  end while
19  return($x^*$);
end C-GRASP;
```

Fig. 7. Pseudo-code for C-GRASP.

improvement phases are then called sequentially in lines 8 and 9, respectively. The solution returned from the local improvement procedure is compared against the current best solution in line 10. If the returned solution has a smaller objective value than the current best solution, then, in lines 11–12, the current best solution is updated with the returned solution. In line 14, if the variables Impr_C and Impr_L are still set to `false`, then the grid density is increased by halving h, in line 15. While variable Impr_C is `false` upon return from the construction procedure if and only if no improvement is made in the construction phase, the Impr_L variable is `false` on return from the local improvement procedure if and only if the input solution to local improvement is determined to be an *h-local minimum*. We increase the grid density at this stage because repeating the construction procedure with the same grid density will not improve the solution. This allows C-GRASP to start with a coarse discretization and adaptively increase the density as needed, thereby intensifying the search in a more dense discretization when a good solution has been found. The best solution found, at the time the stopping criteria are satisfied, is returned.

Table III gives the minimum, 1st quartile, median, mean, 3rd quartile and maximum times (in seconds), as well as the standard deviation of the running times spent to find each optimal (or best know) solution for each problem. According to Table III, on average C-GRASP heuristic found the solutions in less than 12.08, $1,009.00$, 0.10, 604.50, 3.43 seconds for g01, g02, g03, g04, and g05 problems, respectively. In the worst (maximum) case, C-GRASP found the solutions in less than 19.32, $1,881.00$, 0.27, $5,472.00$, and 5.04 seconds, respectively.

Although on instances g01, g03, and g05, the performance of C-GRASP heuristic was a little better than the performance of BRKGA algorithm; the smallest, average, 50% (median), and 70% (3rd quartile) of BRKGA running times were far less than those reported in [Facó et al., 2013] for the C-GRASP heuristic on the two most difficult instances from test suite: g02 and g04 problems.

V. Concluding remarks

In this paper, we present the BRKGA heuristic for finding approximate solutions for continuous global optimization problems subject to box and nonlinear constraints. We illustrate the approach using five challenging problems from CEC2006 benchmark [Liang et al., 2006]. The promising results shown here illustrate the potential of BRKGA for nonlinearly-constrained global optimization problems.

Acknowledgment

The research of R.M.A Silva was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq), the Foundation for Support of Research of the State of Minas Gerais (FAPEMIG), the Brazilian Coordination for the Improvement of Higher Education Personnel (CAPES), the Office for Research and Graduate Studies of the Federal University of Pernambuco (PROPESQ), and the Foundation for Support of Science and Technology of the State of Pernambuco (FACEPE).

References

R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. *J. of Heuristics*, 8:343–373, 2002.

R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. TTTPLOTS: A perl program to create time-to-target plots. *Optimization Letters*, 2006.

J.C. Bean. Genetic Algorithms and Random Keys for Sequencing and Optimization. *ORSA J. on Computing*, 6:154–160, 1994.

M. Ericsson, M.G.C. Resende, and P.M. Pardalos. A genetic algorithm for the weight setting problem in OSPF routing. *J. of Combinatorial Optimization*, 6:299–333, 2002.

Jõao L. Facó, Mauricio G.C. Resende, and Ricardo M.A. Silva. Continuous grasp for nonlinearly-constrained global optimization. In *Congreso Latino-Iberoamericano de Investigación Operativa / Simpósio Brasileiro de Pesquisa Operacional*, CLAIO/SBPO 2013. Sociedade Brasileira de Pesquisa Operacional, 2013.

C.A. Floudas and P.M. Pardalos. *A collection of test problems for constrained global optimization algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 1990. ISBN 0-387-53032-0.

J.F. Gonçalves and J. Almeida. A hybrid genetic algorithm for assembly line balancing. *J. of Heuristics*, 8:629–642, 2002.

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

D.M. Himmelblau. *Applied nonlinear programming*. McGraw-Hill, 1972. URL http://books.google.com.br/books?id=KMpEAAAAIAAJ.

M.J. Hirsch, C.N. Meneses, P.M. Pardalos, and M.G.C. Resende. Global optimization by continuous GRASP. *Optimization Letters*, 1:201–212, 2007.

M.J. Hirsch, P.M. Pardalos, and M.G.C. Resende. Speeding up continuous grasp. *European Journal of Operational Research*, 205(3):507 – 521, 2010. ISSN 0377-2217. doi: 10.1016/j.ejor.2010.02.009. URL http://www.sciencedirect.com/science/article/pii/S0377221710001141.

W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1981. ISBN 0387105611.

S. Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7:pp., 1999.

J. J. Liang, T. P. Runarsson, E. M. Montes, M. Clerc, P. N. Suganthan, C. A. Coello, and Deb K. Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-Parameter Optimization. Technical report, 2006.

M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

Z. Michalewicz, G. Nazhiyath, and M. Michalewicz. A note on usefulness of geometrical crossover for numerical optimization problems. In *Evolutionary Programming*, pages 305–312, 1996.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. URL http://www.R-project.org/. ISBN 3-900051-07-0.

R.M.A. Silva, M.G.C. Resende, P.M. Pardalos, and M.J. Hirsch. A python/c library for bound-constrained global optimization with continuous grasp. *to appear in Optimization Letters*, 2012.

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.