# Continuous GRASP for nonlinearly-constrained global optimization

João L. Facó
Universidade Federal do Rio de Janeiro
Rio de Janeiro, RJ, Brazil.
faco@ufrj.br

Mauricio G. C. Resende
Algorithms and Optimization Research Department
AT&T Labs Research
Florham Park, NJ 07932 USA.
mgcr@research.att.com

Ricardo M. A. Silva
Centro de Informática
Universidade Federal de Pernambuco
Recife, PE, Brazil.
rmas@cin.ufpe.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 continuous GRASP heuristic 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]).

Keywords: Nonlinear constraints, global optimization, continuous optimization, heuristic, local search, nonlinear programming, GRASP, C-GRASP.

# 1 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 $f : S \to R$. In this paper, we consider the domain $S$ as the intersection between a set of nonlinear constraints and a 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 $i = 1, \ldots, n$, in order to present the C-GRASP heuristic for solving bound-constrained continuous global optimization problems subject to nonlinear constraints:

$$\min\ f(x),\ \ x = (x_1, x_2, \ldots, x_n) \tag{1}$$

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 $x_{n+1}, \ldots, x_{n+q}$:

$$g_i(x_1, \ldots, x_n) + x_{n+i} = 0,\ \ i = 1, \ldots, q, \text{ with } x_k \geq 0,\ \ k = n+1, \ldots, n+q, \tag{5}$$

the original problem can be reduced to the following global optimization problem:

$$\min F(x_1, \ldots, x_{n+q}) = [f(x_1, \ldots, x_n) - f^*]^2 + \sum_{i=1}^{q} [g_i(x_1, \ldots, x_n) + x_{n+i}]^2 + \sum_{j=q+1}^{m} [h_j(x_1, \ldots, x_n)]^2 \tag{6}$$

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 $\ni F(z^*) = 0 \Longrightarrow z^*$ is a global minimizer of problem (1-4).

This paper is organized as follows. C-GRASP heuristic for global optimization problem is described in Section 2. In Section 3, experimental results illustrate the effectiveness of C-GRASP for global optimization with nonlinear constraints. Concluding remarks are made in Section 4.

# 2 Continuous GRASP

Continuous GRASP is a method for finding good quality solutions to bound-constrained global optimization problems. A pseudo-code for C-GRASP is shown in Figure 1. The procedure takes as input the problem dimension $n$, lower and upper bound vectors $\ell$ and $u$, the objective function $f(\cdot)$, as well as the parameters $h_s$, $h_e$, and $\rho_{lo}$. As described in Hirsch et al. [2010], parameters $h_s$ and $h_e$ define, respectively, the initial and final grid discretization densities, while parameter $\rho_{lo}$ specifies the portion of the neighborhood of the current solution that is searched during the local improvement phase.

```
procedure C-GRASP(n, ℓ, u, f(·), h_s, h_e, ρ_lo)
1     f* ← ∞;
2     while stopping criteria not met do
3           x ← UnifRand(ℓ, u);
4           h ← h_s;
5           while h ≥ h_e do
6                 Impr_C ← false;
7                 Impr_L ← false;
8                 [x, Impr_C] ← ConstructGreedyRandomized(x, f(·), n, h, ℓ, u, Impr_C);
9                 [x, Impr_L] ← LocalImprovement(x, f(·), n, h, ℓ, u, ρ_lo, Impr_L);
10                if f(x) < f* then
11                      x* ← x;
12                      f* ← f(x);
13                end if
14                if Impr_C = false and Impr_L = false then
15                      h ← h/2;     /* make grid more dense */
16                end if
17          end while
18    end while
19    return(x*);
end C-GRASP;
```

Figure 1: Pseudo-code for C-GRASP.

Line 1 of the pseudo-code initializes the objective function value $f^*$ of the best solution found to infinity. Since C-GRASP is a multi-start procedure, it is continued indefinitely, until one or more stopping criteria are satisfied in line 2. These stopping criteria could be based, for example, on the total number of function evaluations, the number of major iterations, or a target solution quality. The current implementation of the `libcgrpp` library [Silva et al., 2012] can use one of two stopping criteria: (1) stop when the number of outer iterations (loop from line 2 to line 18 in the pseudocode) reaches a specified value; (2) stop when the optimality gap

$$GAP = |f(x) - f(x^*)| \leq \begin{cases} \varepsilon & \text{if } f(x^*) = 0 \\ \varepsilon \cdot |f(x^*)| & \text{if } f(x^*) \neq 0, \end{cases} \tag{9}$$

where $x$ is the current best solution found by the heuristic and $x^*$ is a known global minimum solution, or the best known solution 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 $f(x^*)$.

Each time the stopping criteria of line 2 are not satisfied, another iteration takes place (lines 3–17). During each iteration, the initial solution $x$ is set, in line 3, to a random point distributed uniformly over the $n$-dimensional box defined by $\ell$ and $u$. Parameter $h$, which controls the discretization density of the search space, is re-initialized to $h_s$ in line 4. The construction and local 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 better 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 variables $\texttt{Impr}_\texttt{C}$ and $\texttt{Impr}_\texttt{L}$ are `false`, then the grid density is increased by halving $h$, in line 15. The variable $\texttt{Impr}_\texttt{C}$ (resp. $\texttt{Impr}_\texttt{L}$) is `false` upon return from the construction (resp. local improvement) procedure if and only if no improvement is made in the construction (resp. local improvement) procedure. The grid density is increased 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 with a more dense discretization when no improvement has been found. The best solution found, at the time the

stopping criteria are satisfied, is returned.

The *construction procedure* is shown in Figure 2. It takes as input a solution vector $x$. Initially, the procedure allows all coordinates of $x$ to change (i.e. they are called *unfixed*). In turn, in line 10 of the pseudo-code, if `ReUse` is `false`, a discrete line search is performed in each unfixed coordinate direction $i$ of $x$ with the other $n-1$ coordinates of $x$ held at their current values. Consider the line search in direction $e^i$, where vector $e^i$ has zeros in all components except the $i$-th, where it has value one. The objective function is evaluated at points $x+k\cdot h\cdot e^i$ for $k=0,1,-1,2,-2,\ldots$ such that $l_i \leq x_i+k\cdot h\leq u_i$. Let $k^*$ a the value of $k$ that minimizes $f(x+k\cdot h\cdot e^i)$ subject to $l_i \leq x_i+k\cdot h\leq u_i$. In lines 10 and 11 of the pseudo-code, the value $z_i = x_i+k^*\cdot h$, for the $i$-th coordinate, that minimizes the objective function, together with the objective function value $g_i$, are saved. In line 11, $\breve{x}^i$ denotes $x$ with the $i$-th coordinate set to $z_i$.

```
procedure ConstructGreedyRandomized(x, f(·), n, h, ℓ, u, Impr_C)
1     UnFixed ← {1,2,...,n};
2     α ← UnifRand(0,1);
3     ReUse ← false;
4     while UnFixed ≠ ∅ do
5           g ← +∞;
6           ḡ ← −∞;
7           for i = 1,...,n do
8                 if i ∈ UnFixed then
9                       if ReUse = false then
10                            z_i ← LineSearch(x,h,i,n, f(·),ℓ,u);
11                            g_i ← f(x̆^i);
12                      end if
13                      if g > g_i then g ← g_i;
14                      if ḡ < g_i then ḡ ← g_i;
15                end if
16          end for
17          RCL ← ∅;
18          Threshold ← g + α·(ḡ − g);
19          for i = 1,...,n do
20                if i ∈ UnFixed and g_i ≤ Threshold then
21                      RCL ← RCL ∪ {i};
22                end if
23          end for
24          j ← RandomlySelectElement(RCL);
25          if x_j = z_j then
26                ReUse ← true;
27          else
28                x_j ← z_j;
29                ReUse ← false;
30                Impr_C ← true;
31          end if
32          UnFixed ← UnFixed \ {j};      /* Fix coordinate j. */
33    end while
34    return(x,Impr_C);
end ConstructGreedyRandomized;
```

Figure 2: Pseudo-code for C-GRASP construction phase.

After looping through all unfixed coordinates (lines 7 to 16), in lines 17 to 23 a restricted candidate list (RCL) is formed containing the unfixed coordinates $i$ whose $g_i$ values are less than or equal to $g+\alpha\cdot(\bar{g}-g)$, where $\bar{g}$ and $g$ are, respectively, the maximum and minimum $g_i$ values over all unfixed coordinates of $x$, and $\alpha\in[0,1]$ is chosen uniformly at random in line 2. In line 24, a coordinate is chosen at random from the RCL, say coordinate $j\in$ RCL. Line 25 checks whether $x_j$

and $z_j$ are equal. If so, line 26 sets `ReUse` to the value `true`. Otherwise, in lines 28 to 30, `ReUse` is set to `false`, $\texttt{Impr}_\texttt{C}$ is set to `true`, and $x_j$ is set to equal $z_j$. Finally, in line 32, the coordinate $j$ of $x$ is fixed, by removing $j$ from the set `UnFixed`. The above procedure is continued until all of the $n$ coordinates of $x$ have been fixed. At that stage, $x$ and $\texttt{Impr}_\texttt{C}$ are returned from the construction procedure.

From a given input point $x \in R^n$, the *local improvement procedure* 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 $\bar{x} \in R^n$ be the current solution and $h$ be the current grid discretization parameter. Define

$$S_h(\bar{x}) = \{x \in S \mid \ell \leq x \leq u,\ x = \bar{x} + \tau \cdot h,\ \tau \in Z^n\}$$

to be the set of points in $S$ that are integer steps (of size $h$) away from $\bar{x}$. Let

$$B_h(\bar{x}) = \{x \in S \mid x = \bar{x} + h \cdot (x' - \bar{x})/\|x' - \bar{x}\|,\ x' \in S_h(\bar{x}) \setminus \{\bar{x}\}\}$$

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 $B_h(\bar{x})$.

A pseudo-code for the local improvement procedure is given in Figure 3. The procedure starts from a solution $x \in S \subseteq R^n$ found in the construction procedure. The current best local improvement solution $x^*$ is initialized to $x$ in line 1 of the pseudo-code and the current best solution $f^*$ of the local improvement phase is initialized in line 2 as $f(x^*)$. Based on the current value of the discretization parameter $h$ and the number of points in $B_h(x^*)$, the number of grid points is computed in line 3. In line 4, the number of points `PointsToExamine` that can be evaluated is computed by using parameter $\rho_{lo}$, the portion of the neighborhood to be examined. However, different from what is described in Hirsch et al. [2010], in lines 5 to 7 we impose the restriction that the maximum number of points that can be evaluated in any neighborhood be limited to the value of the parameter `MaxPointsToExamine`.

```
procedure LocalImprovement(x, f(·), n, h, ℓ, u, ρ_lo, Impr_L, MaxPointsToExamine)
1     x* ← x;
2     f* ← f(x);
3     NumGridPoints ← ∏_{i=1}^{n} ⌈(u_i − ℓ_i)/h⌉;
4     PointsToExamine ← ⌈ρ_lo · NumGridPoints⌉;
5     if PointsToExamine > MaxPointsToExamine then
6           PointsToExamine ← MaxPointsToExamine;
7     end if
8     NumPointsExamined ← 0;
9     while NumPointsExamined ≤ PointsToExamine do
10          NumPointsExamined ← NumPointsExamined + 1;
11          x ← RandomlySelectElement(B_h(x*));
12          if ℓ ≤ x ≤ u and f(x) < f* then
13                x* ← x;
14                f* ← f(x);
15                Impr_L ← true;
16                NumPointsExamined ← 0;
17          end if
18    end while
19    return(x*, Impr_L);
end LocalImprovement;
```

Figure 3: Pseudo-code for C-GRASP local improvement phase.

Starting at the point $x^*$, in the loop in lines 9–18 the algorithm randomly selects `PointsToExamine` points in $B_h(x^*)$, one at a time. In line 12, if the current point $x$ selected from $B_h(x^*)$ is feasible and

is better than $x^*$, then $x^*$ is set to $x$, $f^*$ is set to $f(x)$, $\texttt{Impr}_\texttt{L}$ is set to `true`, `NumPointsExamined` is reset to zero, and the process restarts with $x^*$ as the starting solution. $\texttt{Impr}_\texttt{L}$ is used to determine whether the local improvement procedure improved the best solution. Local improvement is terminated if an *h-local minimum* solution $x^*$ is found. At that point, $x^*$ and $\texttt{Impr}_\texttt{L}$ are returned from the local improvement procedure.

## 3 Experimental results

All experiments were done using the Python/C library for C-GRASP introduced by Silva et al. [2012] 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 algorithm used for random-number generation is an implementation of the Mersenne Twister algorithm introduced by Matsumoto and Nishimura [1998]. 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], whose properties are described in Table 1, followed by their corresponding mathematical models.

Table 1: 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 number 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 |

`g01`

$$\min 5 * \sum_{i=1}^{4} x_i - 5 * \sum_{i=1}^{4} {x_i}^2 - \sum_{i=5}^{13} x_i$$

subject to:

$$g_1(x) = 2*x_1 + 2*x_2 + x_{10} + x_{11} - 10 \leq 0$$
$$g_2(x) = 2*x_1 + 2*x_3 + x_{10} + x_{12} - 10 \leq 0$$
$$g_3(x) = 2*x_2 + 2*x_3 + x_{11} + x_{12} - 10 \leq 0$$
$$g_4(x) = -8*x_1 + x_{10} \leq 0$$
$$g_5(x) = -8*x_2 + x_{11} \leq 0$$
$$g_6(x) = -8*x_3 + x_{12} \leq 0$$
$$g_7(x) = -2*x_4 + x_5 + x_{10} \leq 0$$
$$g_8(x) = -2*x_6 - x_7 + x_{11} \leq 0$$
$$g_9(x) = -2*x_8 - x_9 + x_{12} \leq 0$$
$$0 \leq x_i \leq 1, i = 1, \ldots, 9)$$

$$0 \leq x_i \leq 100, i = 10, 11, 12)$$
$$0 \leq x_{13} \leq 1$$

g02

$$\min \; - \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2 * \prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

subject to:

$$g_1(x) = 0.75 - \prod_{i=1}^{n} x_i \leq 0$$
$$g_2(x) = \sum_{i=1}^{n} x_i - 7.5 * n \leq 0$$
$$0 < x_i \leq 10, i = 1, \ldots, n$$

g03

$$\min \; -(\sqrt{n})^n \prod_{i=1}^{n} x_i$$

subject to:

$$h_1(x) = \sum_{i=1}^{n} x_i^2 - 1 = 0$$
$$0 < x_i \leq 1, i = 1, \ldots, n$$

g04

$$\min 5.3578547 x_2 + 0.8356891 x_1 x_5 + 37.293239 x_1 - 40792.141$$

subject to:

$$g_1(x) = 85.334407 + 0.0056858 x_2 x_5 + 0.0006262 x_1 x_4 - 0.0022053 x_3 x_5 - 92 \leq 0$$
$$g_2(x) = -85.334407 - 0.0056858 x_2 x_5 - 0.0006262 x_1 x_4 + 0.0022053 x_3 x_5 \leq 0$$
$$g_3(x) = 80.51249 + 0.0071317 x_2 x_5 + 0.0029955 x_1 x_2 + 0.0021813 x_2 - 110 \leq 0$$
$$g_4(x) = -80.51249 - 0.0071317 x_2 x_5 - 0.0029955 x_1 x_2 - 0.0021813 x_2 + 90 \leq 0$$
$$g_5(x) = 9.300961 + 0.0047026 x_3 x_5 + 0.0012547 x_1 x_3 + 0.0019085 x_3 x_4 - 25 \leq 0$$
$$g_6(x) = -9.300961 - 0.0047026 x_3 x_5 - 0.0012547 x_1 x_3 - 0.0019085 x_3 x_4 + 20 \leq 0$$
$$78 \leq x_1 \leq 102, 33 \leq x_2 \leq 45, 27 \leq x_i \leq 45, i = 3, 4, 5$$

g05

$$\min 3x_1 + 0.000001x_1^3 + 2x_2 + (0.000002/3)x_2^3$$

subject to:

$$g_1(x) = -x_4 + x_3 - 0.55 \leq 0$$

$$g_2(x) = -x_3 + x_4 - 0.55 \leq 0$$

$$h_3(x) = 1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) + 894.8 - x_1 = 0$$

$$h_4(x) = 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0$$

$$h_5(x) = 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) + 1294.8 = 0$$

$$0 \leq x_1, x_2 \leq 1200, 0.55 \leq x_3, x_4 \leq 0.55$$

In all five problems, we ran C-GRASP five times (a different starting random number seed for each run from 270001 to 270005) with $h_s = 0.05$, $h_e = 0.0001$, $rho_{lo} = 0.15$, `MaxPointsToExamine` $=$ 1000, and $\varepsilon = 0.0001$. 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 $F(z^*) = 0$. Therefore, according to Equation (9), we say that the heuristic has solved the problem if $GAP \leq \varepsilon$, in this case with $\varepsilon = 0.0001$. In each problem, the heuristic was able to find its optimal (or best known) solution in all five running.

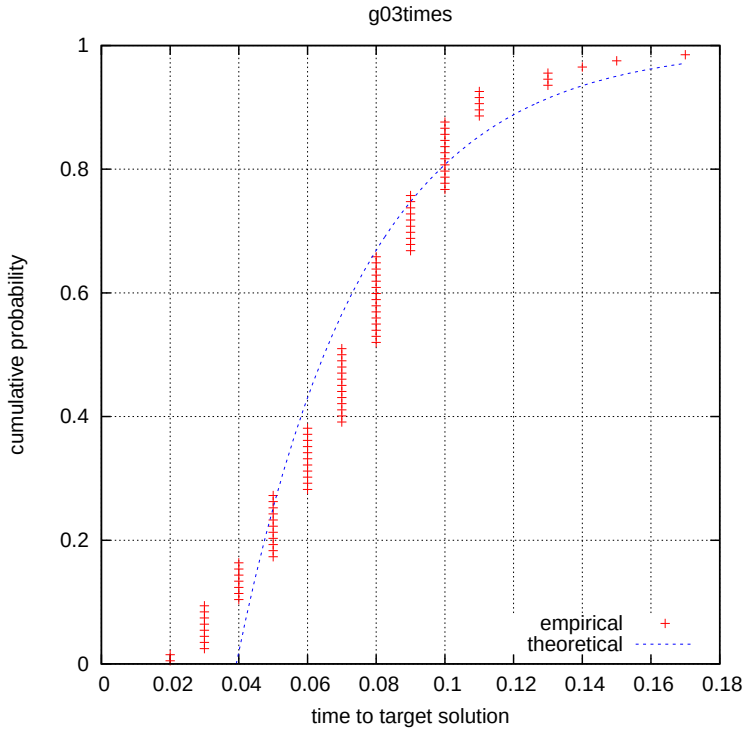


Figure 4: Plot of cumulative probability distribution of C-GRASP running times in seconds for instance g03.

To illustrate the robustness of C-GRASP, we make 100 independent runs of the heuristic on problem g03, recording the time taken to find the best known solution for the instance, and plot its runtime distribution in Figure 4. In other words, Figure 4 shows the time-to-target plot [Aiex

et al., 2002, 2007] (or runtime distributions), where the sorted running times make up the plot. For instance, the running time for 95% of the C-GRASP's runs to find the best-valued solution for problem g03 is 0.145 seconds. Furthermore, the maximum, average and minimum time found were 0.17, 0.0738 and 0.02 seconds, respectively.

Table 2 illustrates the optimum (or best known) value and solution for each problem, $f^* = f(x^*)$ and $x^*$ respectively, as well as the best value $f(x)$ and solution $(x,y)$ found by running C-GRASP heuristic, where $x = (x_1,\ldots,x_n)$ and $y = (x_{n+1},\ldots,x_{n+q})$.

## 4 Concluding remarks

In this paper, we present the C-GRASP 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 C-GRASP for nonlinearly-constrained global optimization problems.

## Acknowledgment

The research of R.M.A Silva was partially done while he was a post-doc scholar at AT&T Labs Research in Florham Park, New Jersey, and 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, Brazil (FAPEMIG), Coordination for the Improvement of Higher Education Personnel, Brazil (CAPES), and Foundation for the Support of Development of the Federal University of Pernambuco, Brazil (FADE).

## 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*, 1:201–212, 2007.

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.

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

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.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.

Table 2: The optimum (or best known) value and solution for problems g01 to g05, as well as the best value and solution found by C-GRASP.

| Prob. | obj. function | solution |
|---|---|---|
| g01 | $-15.0000000000$ | $x^* = (1,1,1,1,1,1,1,1,1,3,3,3,1)$ |
| | $-14.999997702130001$ | $x = (0.999647, 0.998861, 0.999421, 0.999783, 0.999686, 0.999693, 0.999707, 0.999779, 0.999793, 2.999396, 3.006272, 3.002863, 0.999619)$<br>$y = (0.000044, 0.000166, 0.000044, 4.997691, 4.984683, 4.992512, 0.000073, 0.007167, 0.003532)$ |
| g02 | $-0.8036191042$ | $x^* = (3.16246061572185, 3.12833142812967, 3.09479212988791, 3.06145059523469, 3.02792915885555, 2.99382606701730, 2.95866871765285, 2.92184227312450, 0.49482511456933, 0.48835711005490, 0.48231642711865, 0.47664475092742, 0.47129550835493, 0.46623099264167, 0.46142004984199, 0.45683664767217, 0.45245876903267, 0.44826762241853, 0.44424700958760, 0.44038285956317)$ |
| | $-0.8035430237930818$ | $x = (3.175996611086086, 3.180627712694781, 3.066388234840301, 3.058944915994798, 3.116091976677606, 3.030529208214273, 2.967431149927409, 2.992169560380718, 0.487215475849875, 0.475759145750601, 0.481337756961271, 0.464250333517413, 0.454804869592191, 0.451165385881657, 0.462439199478225, 0.460109970824757, 0.452138094957842, 0.450923903942073, 0.451218830443182, 0.460481293395772)$<br>$y = (0.011705681293274, 119.860600403830176)$ |
| g03 | $-1.0005001000$ | $x^* = (0.31624357647283069, 0.31624357741433833 9, 0.31624357801234592 7, 0.31624357566401789 5, 0.31624357820552606 6, 0.31624357738855069, 0.31624357547294951 2, 0.31624357716488393 8, 0.31624357815592030 2, 0.31624357614737491 6)$ |
| | $-1.0004249935155028$ | $x = (0.326040, 0.316221, 0.325100, 0.311989, 0.324127, 0.320020, 0.300434, 0.309490, 0.314283, 0.315552)$ |
| g04 | $-30665.5386717834$ | $x^* = (78, 33, 29.9952560256815985, 45, 36.7758129057882073)$ |
| | $-30665.53862992354$ | $x = (78.017484266249554, 33.000188070479744, 29.703472867008955, 42.329819349577370, 38.189032013036609)$<br>$y = (0.000136633334695, 92.066357110170159, 10.863082742903556, 9.136867955779394, 5.057347781868908, 0.000033058014801)$ |
| g05 | $5126.4967140071$ | $x^* = (679.945148297028709, 1026.06697600004691, 0.118876369094410433, -0.39623348521517826)$ |
| | $5126.496782650232$ | $x = (694.214375, 1010.603029, 0.108629, -0.401050)$<br>$y = (0.041907, 1.061108)$ |