GRASP FOR NONLINEAR OPTIMIZATION

C.N. MENESES, P.M. PARDALOS, AND M.G.C. RESENDE

ABSTRACT. We propose a Greedy Randomized Adaptive Search Procedure (GRASP) for solving continuous global optimization problems subject to box constraints. The method was tested on benchmark functions and the computational results show that our approach was able to find in a few seconds optimal solutions for all tested functions despite not using any gradient information about the function being tested. Most metaheuristcs found in the literature have not been capable of finding optimal solutions to the same collection of functions.

1. INTRODUCTION

In this paper, we consider the global optimization problem: Find a point x^* such that $f(x^*) \leq f(x)$ for all $x \in X$, where X is a convex set defined by box constraints in $\mathbb{R}^n$. For instance, minimize $f(x_1,x_2) = 100(x_2 - x_1^2)^2 + (1-x_1)^2$ such that $-2 \leq x_i \leq 2$ for $i = 1,2$.

Many optimization methods have been proposed to tackle continuous global optimization problems with box constraints. Some of these methods use information about the function being examined, e.g. the Hessian matrix or gradient vector. For some functions, however, this type of information can be difficult to obtain. In this paper, we pursue a heuristic approach to global optimization.

Even though several heuristics have been designed and implemented for the problem discussed in this paper (see e.g. [1, 15, 16, 17]), we feel that there is room for yet another one because these heuristics were not capable of finding optimal solutions to several functions in standard benchmarks for continuous global optimization problems with box constraints.

In this paper, we propose a Greedy Randomized Adaptive Search Procedure (GRASP) for this type of global optimization problem. GRASP [3, 2, 13] is a metaheuristic that has, until now, been applied mainly to find good-quality solutions to discrete combinatorial optimization problems [4]. It is a multi-start procedure which, in each iteration, generates a feasible solution using a randomized greedy construction procedure and then applies local search starting at this solution to produce a locally optimal solution.

In Section 2, we briefly describe what a GRASP is and in Section 3, we show how to adapt GRASP for nonlinear optimization problems. In Section 4, computational results are reported and analyzed. Final remarks are given in Section 5.

Date: June 30, 2005.

Key words and phrases. GRASP, nonlinear programming, global optimization, metaheuristic, heuristic.

AT&T Labs Research Technical Report TD-6DUTRG.

```
Input: Problem instance
Output: Feasible solution, if one exists
InputInstance();
Initialize BestSolution;
while stopping condition is not met do
    ConstructGreedyRandomizedSolution(Solution);
    Solution ← LocalSearch(Solution);
    if Solution is better than BestSolution then
        BestSolution ← Solution;
    end
end
return BestSolution;
```

Algorithm 1: A generic GRASP pseudo-code for optimization problems.

2. GRASP

GRASP is an iterative process, with each GRASP iteration consisting of two phases, a construction phase and a local search phase. The best solution is kept as the final solution. A generic GRASP pseudo-code is shown in Algorithm 1. In that algorithm, the problem instance is read, and then a while loop is executed until a stopping condition is satisfied. This condition could be, for example, the maximum number of iterations, the maximum allowed CPU time, or the maximum number of iterations between two improvements.

In Algorithm 2, a typical construction phase is described. In this phase, a feasible solution is constructed, one element at a time. At each iteration, an element is chosen based on a greedy function. To this end, a candidate list of elements is kept and the greedy function is used to decide which element is chosen to be added to the partial (yet infeasible) solution. The probabilistic aspect of a GRASP is expressed by randomly selecting one of the best candidates (not necessarily the best one) in the candidate list. The list of best candidates is called the *Restricted Candidate List* of simply RCL. For the local search phase, any local search strategy can be used (see Algorithm 3).

```
Input: Problem Instance
Output: Feasible solution, if one exists
Solution ← ∅;
while solution construction not done do
    RCL ← MakeRCL();
    s ← SelectElementAtRandom(RCL);
    Solution ← Solution ∪ {s};
    AdaptGreedyFunction(s);
end
return Solution;
```

Algorithm 2: `ConstructGreedyRandomizedSolution` procedure.

Input: Solution S, Neighborhood $N(S)$
Output: Solution S^*
$S^* \leftarrow S$;
while S^* *is not locally optimal* **do**
  Generate a solution $S \in N(S^*)$;
  if S *is better than* S^* **then**
    $S^* \leftarrow S$;
  end
end
return S^*;

Algorithm 3: `LocalSearch` procedure.

3. GRASP for Nonlinear Optimization Problems

In this section, we describe how to define the construction and local search phases of a GRASP for solving nonlinear optimization problems subject to box constraints. It is assumed that a minimization problem is defined as follows:

$$\min f(x)$$

subject to

$$l_i \leq x_i \leq u_i, \ \forall \ i = 1,2,\ldots,n,$$

where $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$, and l_i and u_i are lower and upper bounds for the values of the variable x_i, respectively.

Since the main contribution of this paper is algorithmic, we describe all algorithms in detail. Algorithm 4 describes our method. The input is a function name given in the column *name* in Tables 1 and 2. In the following paragraphs we explain the steps in that algorithm.

Given the function name (`FuncName`), vectors l and u are initialized according to the last column in Tables 1 and 2. The initial solution (y) is constructed as $y[i] = (l[i] + u[i])/2$ for $i = 1,2,\ldots,n$.

The local search procedure is applied only for problems with at most 4 variables. The reason for this is explained in the computational results section.

Parameter `MaxNumIterNoImprov` controls the grid density (h). The GRASP starts with a given value for h and when the number of iterations with no improvement achieves the value `MaxNumIterNoImprov`, then the value of h is reduced to its half, and the process is restarted. This works as an intensification, forcing the GRASP to search for solutions in a smaller region of the solution space. Though simple, this scheme was important to enhance the performance of the method. We note that most methods for continuous optimization that use the artifice of discretizing the solution space, use a fixed value for h (usually $h = 0.01$). In the GRASP, we start with $h = 1$, and then when necessary, the value of h is reduced. With this, we can look for solutions in a more sparse grid in the beginning and then make the grid denser when we have found a good solution.

Algorithms 5, 6, 7, and 8 describe in detail the procedures used in the GRASP.

The construction phase (Algorithm 5) works as follows. Given a solution vector, $y = (y[1], y[2], \ldots, y[n])$, a line search is performed at coordinate i, $i = 1,2,\ldots,n$, with the $n-1$ other coordinate values fixed. Then, the value for coordinate i that minimizes the function

is kept (see $z[i]$ in Algorithm 5). After performing the line search for each coordinate of y, we select at random a coordinate that has function value (see $g[i]$ in the algorithm) less than or equal to $\texttt{min} + \alpha * (\texttt{max} - \texttt{min})$, where `min` and `max` are, respectively, the minimum and maximum values achieved by the function when executing the line search at each coordinate of y, and α is a user defined parameter ($0 \leq \alpha \leq 1$).

The reason we choose a coordinate using the above strategy is to guarantee the randomness in the construction phase. Then, the solution vector y is updated and the chosen coordinate is eliminated from consideration.

Algorithm 7 shows the line search. The variable `index` in that algorithm represents the coordinate being examined, and the variable $\texttt{xi}_{best}$ keeps track of the best value for that coordinate. Algorithm 6 checks if a given vector $x \in \mathbb{R}^n$ satisfies the box constraints.

In the local search phase (Algorithm 8), a set of directions is examined. Since the functions in our problem can be very complicated, it may not be possible to efficiently compute the gradient of those functions. So, we need a general way to create search directions given a point in $\mathbb{R}^n$.

We use an simple way to generate a set of directions. It depends on the number of variables in the problem. This set of directions is indicated by $D(x)$ in the local search procedure. If the number of variables (n)

- is 1, then $D(x) = \{(-1),(1)\}$;
- is 2, then $D(x) = \{(1,0),(0,1),(-1,0),(0,-1),(1,1),(-1,1),(1,-1),(-1,-1)\}$;
- is 3, then $D(x) = \{\ (1,0,0),\ (0,1,0),\ (0,0,1),\ (-1,0,0),\ (0,-1,0),\ (0,0,-1),\ (1,1,0),\ (1,0,1),\ (0,1,1),\ (-1,1,0),\ (-1,0,1),\ (1,-1,0),\ (0,-1,1),\ (1,0,-1),\ (0,1,-1),\ (1,1,1),\ (-1,1,1),\ (1,-1,1),\ (1,1,-1),\ (1,-1,-1),\ (-1,-1,-1),\ (0,-1,-1),\ (-1,0,-1),\ (-1,-1,0),\ (-1,-1,1),\ (-1,1,-1)\}$.

We note that the number of directions constructed in this way is given by $3^n - 1$. So, for $n = 4$, $D(x)$ would have 80 directions.

The $\texttt{EvaluateFunction}(x, \texttt{FuncName})$ procedure receives as parameters a vector $x \in \mathbb{R}^n$ and the `FuncName`, and returns the corresponding function value at point x.

4. Computational Experiments

In this section, we present the computational experiments carried out with the GRASP described in Section 3. First, we define the benchmark functions used in the tests. Then, in Subsection 4.2, the results obtained by the GRASP are discussed.

```
Input: FuncName, MaxNumIterNoImprov
Output: Feasible solution (BestSolution)
Initialization(FuncName, n, l, u);
for i ← 1 to n do
    y[i] ← (l[i] + u[i])/2;
end
h ← 1;
SolValue ← +∞;
BestSolValue ← +∞;
for iter = 1 to MaxIter do
    ConstructGreedyRandomizedSolution(x, FuncName, n, h, l, u, seed, α);
    if n ≤ 4 then
        LocalSearch(x, xlocalopt, n, FuncName, l, u, h);
        SolValue ← EvaluateFunction(xlocalopt, FuncName);
    else
        SolValue ← EvaluateFunction(x, FuncName);
    end
    if SolValue < BestSolValue then
        BestSolution ← xlocalopt;
        x ← xlocalopt;
        BestSolValue ← SolValue;
        NumIterNoImprov ← 0;
    else
        NumIterNoImprov ← NumIterNoImprov + 1;
    end
    if NumIterNoImprov ≥ MaxNumIterNoImprov then
        /* makes grid more dense */;
        h ← h/2;
        NumIterNoImprov ← 0;
    end
end
return BestSolution;
```

Algorithm 4: GRASP for nonlinear programming.

4.1. **Instances and test environment.** To test the GRASP, we use functions that appear in [7, 8, 9, 10, 14, 16]. These functions are benchmarks commonly used to evaluate unconstrained optimization methods. Tables 1 and 2 list the test functions.

All tests were run in double precision on a Pentium 4 CPU with clock speed of 2.80 GHz and 512 MB of RAM, under MS Windows XP. All algorithms were implemented in the C++ programming language and compiled with GNU g++. CPU times were computed using the function `getusage(RUSAGE_SELF)`.

The algorithm used for random-number generation is an implementation of the multiplicative linear congruential generator [12], with parameters 16807 (multiplier) and $2^{31}-1$ (prime number).

For functions with more than 4 variables, namely F25 through F31, the maximum number of iterations `MaxIter` in Algorithm 4) was set to 10. For all other functions, this parameter was set to 200. Parameter `MaxNumIterNoImprov` in Algorithm 4 was set to 20.

```
Input: x, FuncName, n, h, l, u, α
Output: New solution x
for i = 1 to n do
    y[i] ← x[i];
end
S ← {1, 2, ..., n};
while S ≠ ∅ do
    min ← +∞;
    max ← −∞;
    for each i ∈ S do
        g[i] ← Minimize(y, h, i, n, FuncName, l, u, xi_best);
        z[i] ← xi_best;
        if min > g[i] then
            min ← g[i];
        end
        if max < g[i] then
            max ← g[i];
        end
    end
    RCL ← ∅;
    for each i ∈ S do
        if g[i] ≤ min + α * (max − min) then
            RCL ← RCL ∪ {i};
        end
    end
    j ← RandomlySelectElement(RCL);
    x[j] ← z[j];
    y[j] ← z[j];
    S ← S \ {j};
end
return x ;
```

Algorithm 5: Procedure `ConstructGreedyRandomizedSolution` constructs a greedy randomized solution.

```
Input: x, l, u, n
Output: feas (true or false)
feas ← true;
i ← 1;
while i ≤ n and feas = true do
    if x[i] < l[i] or x[i] > u[i] then
        feas ← false;
    end
    i ← i + 1;
end
return feas;
```

Algorithm 6: Procedure `Feasible` checks if box constraints are satisfied by x.

Input: $y, h, \texttt{index}, n, \texttt{FuncName}, l, u$
Output: minimizer $\texttt{xi}_{best}$
for $i \leftarrow 1$ **to** n **do**
  $t[i] \leftarrow y[i]$;
end
$t[\texttt{index}] \leftarrow l[\texttt{index}]$;
$\texttt{min} \leftarrow +\infty$;
while $t[\texttt{index}] \leq u[\texttt{index}]$ **do**
  $\texttt{value} \leftarrow \texttt{EvaluateFunction}(t, \texttt{FuncName})$;
  if $\texttt{min} > \texttt{value}$ **then**
    $\texttt{min} \leftarrow \texttt{value}$;
    $\texttt{xi}_{best} \leftarrow t[\texttt{index}]$;
  end
  $t[\texttt{index}] \leftarrow t[\texttt{index}] + h$;
end

Algorithm 7: Minimize.

Input: $x, \texttt{xlocalopt}, n, \texttt{FuncName}, l, u, h$
Output: Local optimal xlocalopt
Generate $D(x)$;
$D'(x) \leftarrow D(x)$;
$\texttt{xlocalopt} \leftarrow x$;
$\texttt{xlocaloptValue} \leftarrow \texttt{EvaluateFunction}(x, \texttt{FuncName})$;
repeat
  $\texttt{Improved} \leftarrow \texttt{false}$;
  while $D(\texttt{xlocalopt}) \neq \emptyset$ **and** $\texttt{Improved} = \texttt{false}$ **do**
    select direction $d \in D(\texttt{xlocalopt})$;
    $D \leftarrow D \setminus \{d\}$;
    $\texttt{xprime} \leftarrow \texttt{xlocalopt} + h * d$;
    $\texttt{xprimeValue} \leftarrow \texttt{EvaluateFunction}(\texttt{xprime}, \texttt{FuncName})$;
    if $\texttt{Feasible}(\texttt{xprime}, l, u, n) = \texttt{true}$ **and**
    $\texttt{xprimeValue} < \texttt{xlocaloptValue}$ **then**
      $\texttt{xlocalopt} \leftarrow \texttt{xprime}$;
      $\texttt{xlocaloptValue} \leftarrow \texttt{xprimeValue}$;
      $\texttt{Improved} \leftarrow \texttt{true}$;
      $D(\texttt{xlocalopt}) \leftarrow D'(x)$;
    end
  end
until $\texttt{Improved} = \texttt{false}$;

Algorithm 8: LocalSearch.

4.2. **Experimental Analysis.** In our implementation, the GRASP uses the local search procedure only for functions with at most 4 variables. The reason for this is that for functions with more than 4 variables, the local search becomes too time consuming. Therefore, when the GRASP is used for solving problems with more than 4 variables, it uses only the construction procedure. We emphasis that just the construction procedure was able to find optimal solutions for all functions tested in this paper. It is by itself a method for

TABLE 1. Benchmark functions. Functions F1 through F19 were tested in [16] and are benchmark for testing nonlinear unconstrained optimization methods. F20 and F21 appear in [7]. F22 and F23 are from [14]. Data for F4 and F5 are given in Table 4.1.

Name	Function	Domain
F1	$[1+(x_1+x_2+)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)]$. $[30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)]$	$-2 \le x_i \le 2$
F2	$(x_1^2+x_2-11)^2+(x_1+x_2^2-7)^2$	$-6 \le x_i \le 6$
F3	$100(x_2-x_1^2)^2+(1-x_1)^2$	$-2 \le x_i \le 2$
F4	$-\sum_{i=1}^{5}[(x-a_i)^T(x-a_i)+c_i]^{-1}$	$0 \le x_i \le 10$
F5	$-\sum_{i=1}^{10}[(x-a_i)^T(x-a_i)+c_i]^{-1}$	$0 \le x_i \le 10$
F6	$(x_1+10x_2)^2+5(x_3-x_4)^2+(x_2-2x_3)^4+10(x_1-x_4)^4$	$-3 \le x_i \le 3$
F7	$(4-2.1x_1^2+x_1^4/3)x_1^2+x_1x_2+(-4+4x_2^2)x_2^2$	$-3 \le x_1 \le 3, -2 \le x_2 \le 2$
F8	$\{\sum_{i=1}^{5} i\cos[(i+1)x_1+i]\}.\{\sum_{i=1}^{5} i\cos[(i+1)x_2+i]\}$	$-10 \le x_i \le 10$
F9	$\frac{1}{2}\sum_{i=1}^{2}(x_i^4-16x_i^2+5x_i)$	$-20 \le x_i \le 20$
F10	$\frac{1}{2}\sum_{i=1}^{3}(x_i^4-16x_i^2+5x_i)$	$-20 \le x_i \le 20$
F11	$\frac{1}{2}\sum_{i=1}^{4}(x_i^4-16x_i^2+5x_i)$	$-20 \le x_i \le 20$
F12	$0.5x_1^2+0.5[1-\cos(2x_1)]+x_2^2$	$-5 \le x_i \le 5$
F13	$10x_1^2+x_2^2-(x_1^2+x_2^2)^2+10^{-1}(x_1^2+x_2^2)^4$	$-5 \le x_i \le 5$
F14	$10^2x_1^2+x_2^2-(x_1^2+x_2^2)^2+10^{-2}(x_1^2+x_2^2)^4$	$-5 \le x_i \le 5$
F15	$10^3x_1^2+x_2^2-(x_1^2+x_2^2)^2+10^{-3}(x_1^2+x_2^2)^4$	$-20 \le x_i \le 20$
F16	$10^4x_1^2+x_2^2-(x_1^2+x_2^2)^2+10^{-4}(x_1^2+x_2^2)^4$	$-20 \le x_i \le 20$
F17	$x_1^2+x_2^2-\cos(18x_1)-\cos(18x_2)$	$-5 \le x_i \le 5$
F18	$[x_2-(5.1x_1^2)/(4\pi^2)+5x_1/\pi-6]^2+10[1-1/(8\pi)]\cos(x_1)+10$	$-20 \le x_i \le 20$
F19	$-\{\sum_{i=1}^{5}\sin[(i+1)x_1+i]\}$	$-20 \le x_1 \le 20$
F20	$e^{u^2}+[\sin(4x_1-3x_2)]^4+0.5(2x_1+x_2-10)^2, \quad u=0.5(x_1^2+x_2^2-25)$	$0 \le x_i \le 6$
F21	$x_1^6-15x_1^4+27x_1^2+250$	$-5 \le x_1 \le 5$
F22	$100(x_2-x_1^2)^2+(1-x_1)^2+90(x_4-x_3^2)^2+(1-x_3)^2+$ $10.1[(x_2-1)^2+(x_4-1)^2]+19.8(x_2-1)(x_4-1)$	$-3 \le x_i \le 3$
F23	$[1.5-x_1(1-x_2)]^2+[2.25-x_1(1-x_2^2)]^2+[2.625-x_1(1-x_2^3)]^2$	$0 \le x_i \le 5$

TABLE 2. Continued benchmark functions. F24 through F28 are from [9]. Note that F25, F26, F27 and F28 are the same quadratic function, but with different number of variables. F29 through F31 appear in [10]. F32 is from [8].

Name	Function	Domain
F24	$\sum_{i=1}^{10}(e^{-0.2i}+2e^{-0.4i}-x_1e^{-0.2x_2i}-x_3e^{-0.2x_4i})^2$	$0 \le x_i \le 4$
F25	$\sum_{i=1}^{10}\frac{x_i^2}{2^{i-1}}+\sum_{i=2}^{10}\frac{x_ix_{i-1}}{2^i}$	$-20 \le x_i \le 7$
F26	$\sum_{i=1}^{20}\frac{x_i^2}{2^{i-1}}+\sum_{i=2}^{20}\frac{x_ix_{i-1}}{2^i}$	$-20 \le x_i \le 7$
F27	$\sum_{i=1}^{30}\frac{x_i^2}{2^{i-1}}+\sum_{i=2}^{30}\frac{x_ix_{i-1}}{2^i}$	$-20 \le x_i \le 7$
F28	$\sum_{i=1}^{40}\frac{x_i^2}{2^{i-1}}+\sum_{i=2}^{40}\frac{x_ix_{i-1}}{2^i}$	$-20 \le x_i \le 7$
F29	$\sum_{i=1}^{10}x_i^2$	$-10 \le x_i \le 7$
F30	$\sum_{i=1}^{10}\lfloor x_i^2+0.5\rfloor^2$	$-10 \le x_i \le 7$
F31	$-20exp(-0.2\sqrt{\frac{1}{10}\sum_{i=1}^{10}x_i^2})-exp(\frac{1}{10}\sum_{i=1}^{10}\cos(2\pi x_i))+20+e$	$-10 \le x_i \le 20$
F32	$\sin(x_1)+\sin(10x_1/3)+\log_{10}(x_1)-0.84x_1$	$0.1 \le x_1 \le 6$

TABLE 3. Data for functions F4 and F5.

i	a_i				c_i	i	a_i				c_i
1	4	4	4	4	0.1	6	2	9	2	9	0.6
2	1	1	1	1	0.2	7	5	5	3	3	0.3
3	8	8	8	8	0.2	8	8	1	8	1	0.7
4	6	6	6	6	0.4	9	6	2	6	2	0.5
5	3	7	3	7	0.4	10	7	3.6	7	3.6	0.5

finding near optimal solutions for continuous optimization problems with box constraints. However, for some problems, the inclusion of local search, in addition to the construction, reduced the CPU time to find solutions.

Table 4.2 shows the results for the GRASP. In this table, the first column shows the function name, the second and third columns give the solution value for the solution found by GRASP, and the CPU time (in seconds) to find that solution, the fourth column shows the global optimum value as reported in the literature, and the last column gives the absolute value of the difference between the values in the second and fourth columns. Note that the smaller the value appearing in the last column the better the solution found by our method.

Table 4.2 shows the solution vectors found by GRASP.

As seen in Table 4.2, the solutions found by GRASP are optimal or near-optimal. We remark that increasing the value of the parameter `MaxIter` (maximum number of iterations in Algorithm 4) generally improves the solution found.

TABLE 4. In the column CPU time, zero means that the GRASP executed in less than 1 second.

	Solution		Global Optimum	Absolute Distance
Name	Value	CPU time	Value	Value
F1	3.0	0	3.0	0.0
F2	0.0	0	0.0	0.0
F3	0.0	0	0.0	0.0
F4	-10.1531958	3	-10.1531957	0.0000001
F5	-10.5362837	5	-10.5362836	0.0000001
F6	0.0	0	0.0	0.0
F7	-1.0316280	0	-1.031628	0.0
F8	-186.7295368	1	-186.7309	0.0013632
F9	-78.3323112	1	-78.332331	0.0000219
F10	-117.4984669	3	-117.4984	0.0000669
F11	-156.6646225	6	-156.66466	0.0000375
F12	0.0	0	0.0	0.0
F13	-0.4074610	0	-0.407461	0.0
F14	-18.0586860	0	-18.058697	0.000011
F15	-227.7656948	2	-227.765747	0.0000522
F16	-2429.4146131	2	-2429.414749	0.0001359
F17	-2.0	0	-2.0	0.0
F18	0.3978919	1	0.397887	0.0000049
F19	-3.3728854	1	-3.372897	0.0000116
F20	1.0	0	1.0	0.0
F21	7.0	0	7.0	0.0
F22	0.0	1	0.0	0.0
F23	0.0	0	0.0	0.0
F24	0.0	0	0.0	0.0
F25	0.0	0	0.0	0.0
F26	0.0	1	0.0	0.0
F27	0.0	4	0.0	0.0
F28	0.0	11	0.0	0.0
F29	0.0	0	0.0	0.0
F30	0.0	0	0.0	0.0
F31	0.0	0	0.0	0.0
F32	-5.5344323	0	-5.534	0.0004323

We observed from the experiments that the GRASP needed *at most two* iterations to find optimal solutions for problems with quadratic functions, namely F25 through F30. Figures 1, 2, and 3 show the progress of the GRASP for the functions that took at least 2 iterations to converge to the values on the second column of Table 4.2.

TABLE 5. Solution vectors found by GRASP.

Name	Solution Vector
F1	(0,-1)
F2	(3,2)
F3	(1,1)
F4	(4,4,4,4)
F5	(4,4,4,4)
F6	(0,0,0,0)
F7	(0.089843,-0.712890)
F8	(5.482421, 4.857421)
F9	(-2.90429,-2.90429)
F10	(-2.90429,-2.90429,-2.90429)
F11	(-2.90429,-2.90429,-2.90428,-2.90429)
F12	(0,0)
F13	(0,-1.386718)
F14	(0,-2.609375)
F15	(0,-4.701171)
F16	(0,-8.394531)
F17	(0,0)
F18	(3.140625,2.275390)
F19	(18.412109)
F20	(3,4)
F21	(-3)
F22	(1,1,1,1)
F23	(3,0.5)
F24	(1,1,2,2)
F25	(0,0,. . .,0)
F26	(0,0,. . .,0)
F27	(0,0,. . .,0)
F28	(0,0,. . .,0)
F29	(0,0,. . .,0)
F30	(0,0,. . .,0)
F31	(0,0,. . .,0)
F32	(5.209375)

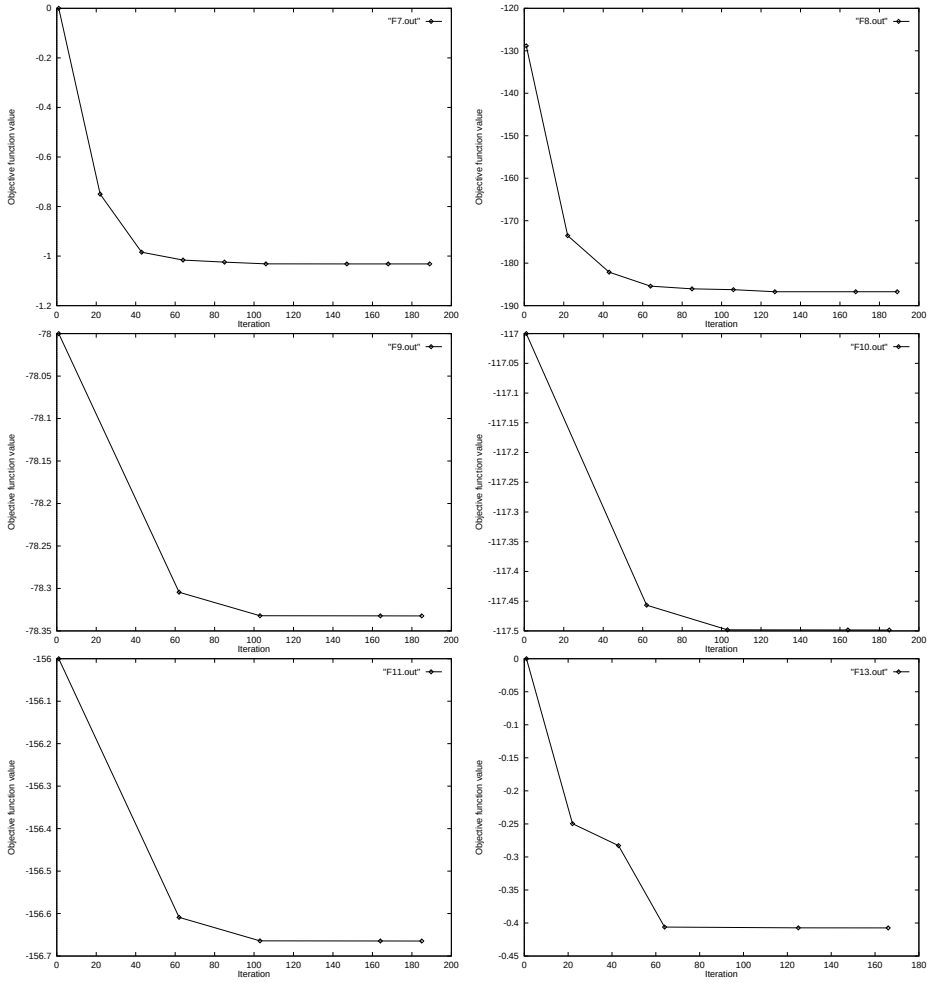

FIGURE 1. GRASP progress for functions F7, F8, F9, F10, F11 and F13.

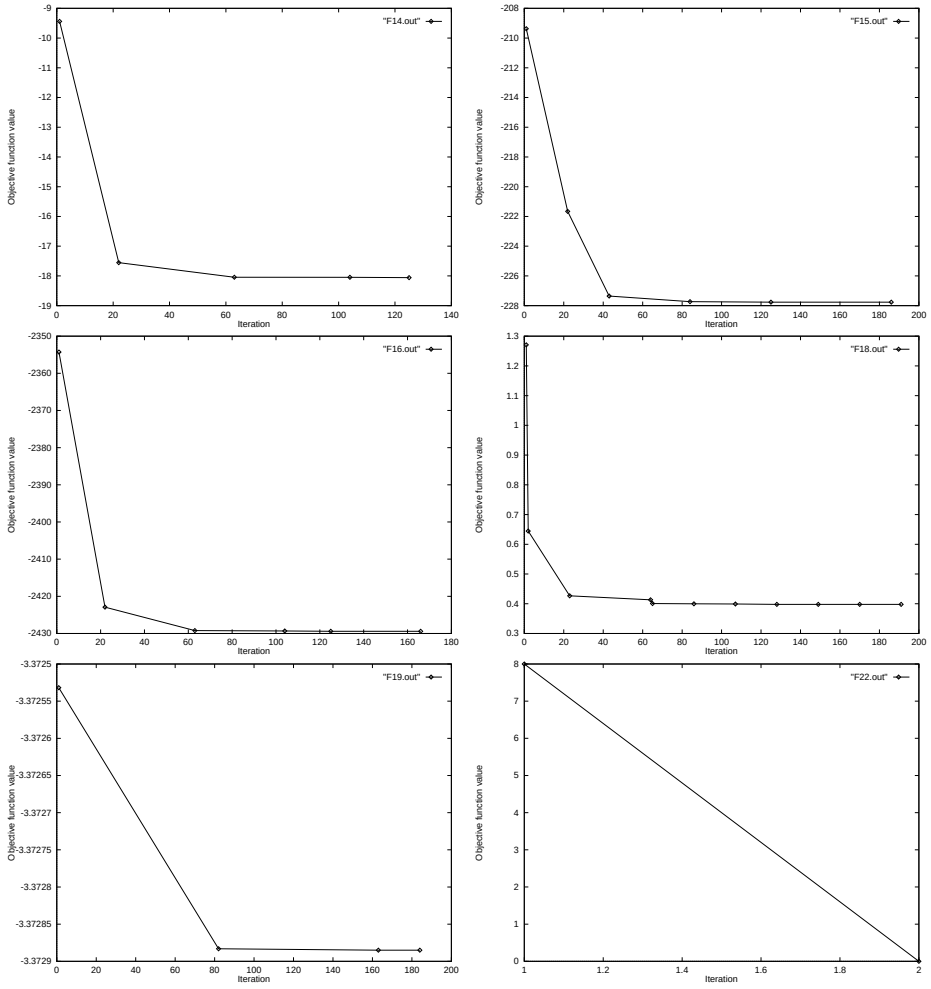

FIGURE 2. GRASP progress for functions F14, F15, F16, F18, F19 and F22.

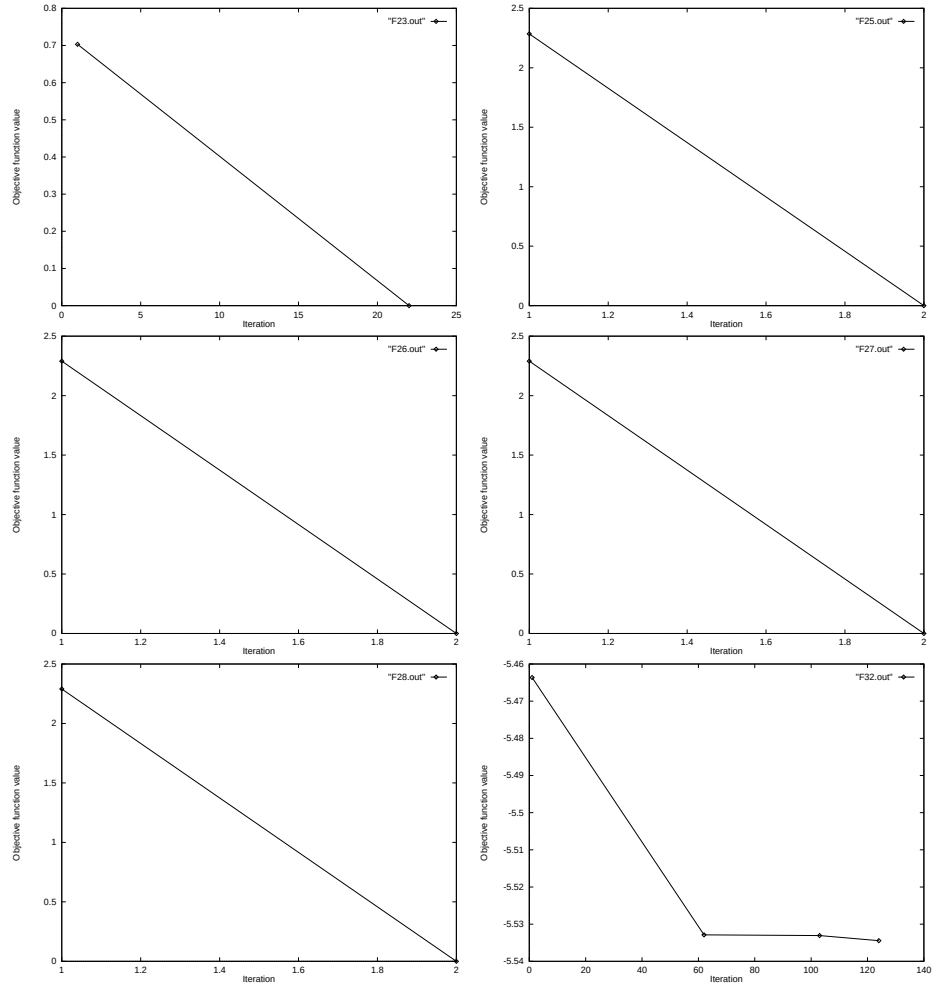

FIGURE 3. GRASP progress for functions F23, F25, F26, F27, F28 and F32.

5. Final Remarks

In this paper, we described a new approach for continuous global optimization problems subject to box constraints. The method does not use any particular information about the objective function and is very simple to implement.

The computational experiments showed that the method is quite promising since it was capable of finding optimal or near-optimal solutions for all tested functions. Other methods do not have similar behavior on the same collection of functions.

Acknowledgments

This work has been partially supported by NSF, NIH and CRDF grants. The research of Claudio Meneses was supported in part by the Brazilian Federal Agency for Higher Education (CAPES) – Grant No. 1797-99-9.

References

[1] J. G. Digalakis and K. G. Margaritis. An experimental study of benchmarking functions for genetic algorithms. *International J. Computer Math.*, 79(4):403–416, 2002.

[2] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[3] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[4] P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and surveys in metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

[5] C. A. Floudas and P. M. Pardalos. A collection of test problems for constrained global optimization algorithms. *Lecture Notes in Computer Science, G. Goos and J. Hartmanis, Eds., Springer Verlag*, 455, 1990.

[6] 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*. Kluwer Academic Publishers, Dordrecht, 1999.

[7] A. A. Goldstein and J. F. Price. On descent from local minima. *Mathematics of Computation*, 25(115):569–574, 1971.

[8] G. H. Koon and A. V. Sebald. Some interesting test functions for evaluating evolutionary programming strategies. *Proc. of the Fourth Annual Conference on Evolutionary Programming*, pages 479–499, 1995.

[9] A. I. Manevich and E. Boudinov. An efficient conjugate directions method without linear minimization. *Submitted to Computer Physics Communications*, 1999.

[10] J. R. McDonnell and D. E. Waagen. An empirical study of recombination in evolutionary search. *Proc. of the Fourth Annual Conference on Evolutionary Programming*, pages 465–478, 1995.

[11] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, 1981.

[12] S. Park and K. Miller. Random number generators: Good ones are hard to find. *Communications of the ACM*, 31:1192–1201, 1988.

[13] M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2002.

[14] D. F. Shanno and K. H. Phua. Minimization of unconstrained multivariate functions. *ACM Transactions on Mathematical Software*, 2(1):87–94, 1976.

[15] A. B. Simoes and E. Costa. Transposition versus crossover: An empirical study. *Proc. of the Genetic and Evolutionary Computation Conference*, pages 612–619, 1999.

[16] T. B. Trafalis and S. Kasap. A novel metaheuristics approach for continuous global optimization. *Journal of Global Optimization*, 23:171–190, 2002.

[17] J.-M. Yang and C. Y. Kao. A combined evolutionary algorithm for real parameters optimization. *Proc. of IEEE International Conference on Evolutionary Computation*, pages 732–737, 1996.

(C.N. Meneses) DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING, UNIVERSITY OF FLORIDA, 303 WEIL HALL, GAINESVILLE, FL 32611
E-mail address, C.N. Meneses: claudio@ufl.edu

(P.M. Pardalos) DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING, UNIVERSITY OF FLORIDA, 303 WEIL HALL, GAINESVILLE, FL 32611
E-mail address, P.M. Pardalos: pardalos@ufl.edu

(M.G.C. Resende) INTERNET AND NETWORK SYSTEMS RESEARCH CENTER, AT&T LABS RESEARCH, 180 PARK AVENUE, ROOM C241, FLORHAM PARK, NJ 07932 USA.
E-mail address, M.G.C. Resende: mgcr@research.att.com