SPEEDING UP CONTINUOUS GRASP

M. J. HIRSCH, P. M. PARDALOS, AND M. G. C. RESENDE

ABSTRACT. Continuous GRASP (C-GRASP) is a stochastic local search metaheuristic for finding cost-efficient solutions to continuous global optimization problems subject to box constraints (Hirsch et al., 2006). Like a greedy randomized adaptive search procedure (GRASP), a C-GRASP is a multi-start procedure where a starting solution for local improvement is constructed in a greedy randomized fashion. In this paper, we describe several improvements that speed up the original C-GRASP and make it more robust. We compare the new C-GRASP with the original version as well as with other algorithms from the recent literature on a set of benchmark multimodal test functions whose global minima are known. Hart's sequential stopping rule (1998) is implemented and C-GRASP is shown to converge on all test problems.

1. INTRODUCTION

Global optimization problems are abundant in the applied sciences [5, 16]. Global optimization seeks a minimum or maximum of a multimodal function over a discrete or continuous domain. In its minimization form, global optimization is stated mathematically as finding a solution $x^* \in S \subseteq \mathbb{R}^n$ such that $f(x^*) \leq f(x)$, $\forall\, x \in S$, where S is some region of $\mathbb{R}^n$ and the multimodal objective function f is defined by $f : S \to \mathbb{R}$. Such a solution x^* is called a *global minimum*.

Recently, several derivative-free metaheuristics for continuous global optimization problems have been proposed [8, 9, 12]. Many of these metaheuristics were originally proposed for discrete combinatorial optimization problems and have been adapted to deal with continuous optimization problems. GRASP, or greedy randomized adaptive search procedures [2, 3], can be included in this group of metaheuristics. It has been applied to a wide range of combinatorial optimization problems [3, 4, 17]. Recently, Hirsch et al. [10] proposed an adaptation of GRASP for continuous global optimization. Continuous GRASP (or simply C-GRASP) was shown to perform well on a set of 14 multimodal test functions, as well as on two difficult real-world applications. In [11], C-GRASP was applied to the registration of sensors in a sensor network.

In this paper, we describe several improvements to the C-GRASP metaheuristic described in Hirsch et al. [10]. These improvements are aimed at speeding up implementations of the algorithm and increasing robustness, while at the same time keeping the overall algorithm simple to implement. This paper is organized as follows. In Section 2, several improvements to C-GRASP are described. Section 3 compares the new improved C-GRASP with the C-GRASP of [10] and three other heuristics on a set of 40 multimodal test functions. We also evaluate the performance of C-GRASP as a multi-start heuristic using the sequential stopping rules of Hart [6]. Concluding remarks are made in Section 4.

Date: September 28, 2006.

Key words and phrases. GRASP, continuous GRASP, global optimization, multimodal functions, continuous optimization, heuristic, stochastic algorithm, stochastic local search, nonlinear programming.

AT&T Labs Research Technical Report TD-6U2P2H..

```
procedure GRASP(Problem Instance)
1       InputInstance();
2       while stopping criteria not met do
3               ConstructGreedyRandomizedSolution(Solution);
4               LocalSearch(Solution);
5               if Solution is better than BestSolution then
6                       UpdateBestSolution(Solution,BestSolution);
7               end if
8       end while
9       return(BestSolution);
end GRASP;
```

FIGURE 1. High-level pseudo-code for GRASP.

```
procedure ConstructGreedyRandomizedSolution(Problem Instance)
1       Solution ← ∅;
2       while Solution construction not done do
3               MakeRCL(RCL);
4               S ← SelectRandomElement(RCL);
5               Solution ← Solution ∪ {S};
6               AdaptGreedyFunction(S);
7       end while
8       return(Solution);
end ConstructGreedyRandomizedSolution;
```

FIGURE 2. High-level pseudo-code for Construction procedure.

```
procedure LocalSearch(Solution, Neighborhood))
1       Solution* ← Solution;
2       while Solution* not locally optimal do
3               Solution ← SelectRandomElement(Neighborhood(Solution*));
4               if Solution better than Solution* then
5                       Solution* ← Solution;
6               end if
7       end while
8       return(Solution*);
end LocalSearch;
```

FIGURE 3. High-level pseudo-code Local Search procedure.

2. IMPROVEMENTS TO CONTINUOUS GRASP

Feo and Resende [2, 3] describe the metaheuristic GRASP as a multi-start local search procedure, where each GRASP iteration consists of two phases, a construction phase and a local search phase. In the construction phase, interactions between greediness and randomization generate a diverse set of good-quality solutions. The local search phase attempts to improve the solutions found by construction. The best solution over all of the multi-start iterations is retained as the final solution. Figures 1–3 provide high-level pseudo-code of the main GRASP algorithm, as well as of the construction and local search phases.

Hirsch et al. [10] describe C-GRASP, an adaptation of GRASP to handle continuous optimization problems. C-GRASP works by discretizing the domain into a uniform grid.

```
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 4. Pseudo-code for C-GRASP.

Both the construction and local improvement phases move along points on the grid. As the algorithm progresses, the grid adaptively becomes more dense. C-GRASP resembles GRASP in that it is a multi-start stochastic search metaheuristic that uses a randomized greedy construction procedure to generate starting solutions for a local improvement algorithm. The main difference is that an iteration of C-GRASP does not consist of a single greedy randomized construction followed by local improvement, but rather a series of construction-local improvement cycles with the output of construction serving as the input of the local improvement, as in GRASP, but unlike GRASP, the output of the local improvement serves as the input of the construction procedure.

In the remainder of this section, we propose some improvements to C-GRASP for solving continuous global optimization problems subject to box constraints. We shall refer to this algorithm as the *new C-GRASP* in contrast to the algorithm described in [10] which we will call the *original C-GRASP*. In Section 2.1, we present the new C-GRASP. Section 2.2 describes the construction procedure. Finally, in Section 2.3, we describe the local improvement procedure. Without loss of generality, we take the domain S to be the hyper-rectangle $S = \{x = (x_1, \ldots, x_n) \in \mathbb{R}^n : \ell \leq x \leq u\}$, where $\ell, u \in \mathbb{R}^n$ such that $\ell \leq u$. The minimization problem considered in this paper therefore becomes: Find $x^* = \text{argmin}\{f(x) \mid \ \ell \leq x \leq u\}$, where $f : \mathbb{R}^n \to \mathbb{R}$, and $\ell, x, u \in \mathbb{R}^n$.

2.1. **Continuous GRASP.** Pseudo-code for the new C-GRASP is shown in Figure 4. 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 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. Since C-GRASP is a multi-start procedure, it is continued indefinitely,

until one or more stopping criteria are satisfied. These stopping criteria could be based, for example, on the total number of function evaluations performed or the time elapsed since the start of the algorithm. Since different implementations of C-GRASP will have different stopping criteria, we list line 2 in general form.

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 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 $\texttt{Impr}_\texttt{C}$ and $\texttt{Impr}_\texttt{L}$ are still set to `false` , then the grid density is increased by halving h, in line 15. As will be seen in Section 2.2, the variable $\texttt{Impr}_\texttt{C}$ is `false` upon return from the construction procedure if and only if no improvement is made in the construction phase. Section 2.3 shows that the $\texttt{Impr}_\texttt{L}$ 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, as we will see in Section 2.2, 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.

The main differences between this new version of C-GRASP and the one in [10] are limited to the input parameters and when the grid density is increased. In [10], the input parameters were `MaxIters`, the number of iterative cycles in each of the multi-start runs, and `MaxNumIterNoImprov`, the number of iterations without improvement until the grid density was increased. The strategy proposed in [10], i.e. based on fixed number of iterative cycles in each of the multi-start iterations and number of iterations without improvement until the grid density was increased, sometimes led to excessively slow convergence. Parameters h_s and h_e in the algorithm proposed in this paper allow one to specify how coarse to start the grid and how fine the grid should be before stopping. As we shall see later in this section, increasing the grid density when the variables $\texttt{Impr}_\texttt{C}$ and $\texttt{Impr}_\texttt{L}$ are still set to `false` in line 14 allows the algorithm to avoid unnecessary iterations with no chance of producing an improved solution.

2.2. **Construction procedure.** In this section, we describe in detail the construction procedure. As stated above, the construction algorithm combines greediness and randomization to produce a diverse set of good-quality solutions from which to start the local improvement phase. The construction algorithm is shown in Figure 5. The input is a solution vector x. To start, the algorithm allows all coordinates of x to change (i.e. they are unfixed). In turn, in line 10 of the pseudo-code, if `ReUse` is `false` , a 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. In lines 10 and 11 of the pseudo-code, the value z_i 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.

After looping through all unfixed coordinates (lines 7–16), in lines 17–23 a restricted candidate list (RCL) is formed containing the unfixed coordinates i whose g_i values are less than or equal to $\underline{g} + \alpha \cdot (\bar{g} - \underline{g})$, where $\bar{g}$ and $\underline{g}$ are, respectively, the maximum and

```
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(x^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 5. Pseudo-code for C-GRASP construction phase.

minimum g_i values over all unfixed coordinates of x, and $\alpha \in [0,1]$ is randomly determined 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 28–30, 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 30, we fix coordinate j of x, by removing j from the set UnFixed. Choosing a coordinate by selecting at random from the RCL ensures both greediness and randomness in the construction phase. 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.

At this point, it is worthwhile to examine two aspects of the construction procedure. The first concerns the need for the ReUse parameter. Suppose that at some stage in the construction procedure, we reach line 25 and determine that $x_j = z_j$. Then, when we had computed lines 7–16 just before this stage, and for each $i \in \texttt{UnFixed} \setminus \{j\}$ we performed a line search along the i-th coordinate, keeping all other coordinates of x fixed at their

present values. Therefore, $\forall\, i \in \texttt{UnFixed} \setminus \{j\}$, the value of x_i is changed during the line search, while the value of x_j remains unchanged. If line 25 is `true`, then the next time around through lines 7–16 with j no longer in `UnFixed`, the j-th coordinate of x will not have changed. Hence, performing a line search in the remaining unfixed coordinates of x will produce the same results as previously. Setting `ReUse` to `true` allows the construction procedure to use the z_i and g_i values computed previously. This has the effect of speeding up the procedure since fewer calls are made to `LineSearch`, leading to fewer function evaluations. Thus, we have the following:

Proposition 2.1. *If, at any stage of the construction procedure (Fig. 5), line 25 is true (i.e. $z_j = x_j$), then the values of z_i and g_i, $\forall\, i \in \texttt{UnFixed} \setminus \{j\}$ will remain valid for the next stage of the procedure.*

The second point worthwhile noting was initially brought up in Section 2.1, when reference was made to halving h. Suppose $\hat{x}$ was input to the construction procedure and the same $\hat{x}$ was output unchanged from the construction procedure. If, using the same discretization value, $\hat{x}$ is again input to the construction procedure, then for the i-th coordinate direction, keeping the other $n-1$ coordinates of $\hat{x}$ fixed, the i-th coordinate that minimizes the objective function is $\hat{x}_i$. Therefore, the first time through lines 7–16 of the construction procedure, z_i will be set equal to $\hat{x}_i$ and g_i will be set equal to $f(\hat{x})$, $\forall\, i = 1,\ldots,n$. Therefore, we have the following:

Proposition 2.2. *Let h be the current grid discretization parameter and $\hat{x}$ be the input solution to the construction procedure. If no improvement to $\hat{x}$ is made in the construction procedure, then $f(\hat{x}) \leq f(x')$, $\forall\, x' \in \{x \mid \ell \leq x \leq u,\ x = \hat{x} + \gamma \cdot h \cdot e_i,\ \gamma \in \mathbb{Z},\ i \in \{1,\ldots,n\}\}$, where e_i is the unit vector with 1 in the i-th coordinate and zeroes elsewhere.*

The main improvement proposed for the construction phase is related to the use of the variable `ReUse` to speed up computations by avoiding unnecessary line searches. We also propose the use of a randomly determined α parameter rather than a fixed parameter. The parameter α controls the size of the `RCL` and therefore determines the mix of greediness and randomness in the construction procedure. Different values of α throughout the run allow some construction phases to be more greedy while others are more random.

2.3. **Local improvement procedure.** C-GRASP 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 6) 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

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

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 $B_h(\bar{x})$.

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

FIGURE 6. Pseudo-code for C-GRASP local improvement phase.

The local improvement procedure is given a starting solution $x \in S \subseteq \mathbb{R}^n$. The current best local improvement solution x^* is initialized to x in line 1. Lines 3 and 4 determine the number of grid points, based on the current value of the discretization parameter h, and the maximum number of points in $B_h(x^*)$ that are to be examined. This number of grid points is defined by the parameter ρ_{lo} which is the portion of the neighborhood which is to be examined. If all of these points are examined and no improving point is found, the current solution x^* is considered an *h-local minimum*.

Starting at the point x^*, in the loop in lines 6–15, the algorithm randomly selects `MaxPointsToExamine` points in $B_h(x^*)$, 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, $\texttt{Impr}_\texttt{L}$ is set to `true`, 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.

The differences between the local improvement procedure in [10] and the one presented here are limited to the neighborhood of the current solution and the number of points in the neighborhood that are examined. Suppose that the current solution is $\bar{x}$. The local improvement procedure in [10] would examine up to `MaxDirToTry` points of the form $\bar{x} + h \cdot \{-1, 0, 1\}^n$, where `MaxDirToTry` is a user-supplied parameter. Hence, only points on the current grid could be examined. However, there is no reason to assume that a local minimum will occur on the current grid. With the local improvement procedure described here, the possible points to be examined are the current grid points projected onto a hyper-sphere of radius h about $\bar{x}$. These points may not be on the current grid. The number of points to examine is a function of the user-defined parameter ρ_{lo} and increases with the dimension of the problem.

3. EXPERIMENTAL RESULTS

We study the performance of the new C-GRASP through three experiments. First, we compare the original C-GRASP described in [10] with the new C-GRASP described in this

TABLE 1. Test problems. Function names in boldface indicates function was used in the comparison of the new and original C-GRASPs.

Function name	Dimension	Function name	Dimension
Beale (BE)	2	Bohachevsky (B_2)	2
Booth (BO)	2	**Branin** (BR)	2
Easom (EA)	2	**Goldstein and Price** (GP)	2
Matyas (M)	2	**Rosenbrock** (R_2)	2
Schwefel (SC_2)	2	**Shubert** (SH)	2
Six-Hump Camelback (CA)	2	Zakharov (Z_2)	2
De Joung (SP_3)	3	**Hartmann** ($H_{3,4}$)	3
Colville (CV)	4	Perm ($P_{4,\frac{1}{2}}$)	4
Perm$_0$ ($P^0_{4,10}$)	4	Power Sum ($PS_{4,\{8,18,44,114\}}$)	4
Shekel ($S_{4,5}$)	4	**Shekel** ($S_{4,7}$)	4
Shekel ($S_{4,10}$)	4	**Rosenbrock** (R_5)	5
Zakharov (Z_5)	5	**Hartmann** ($H_{6,4}$)	6
Schwefel (SC_6)	6	Trid (T_6)	6
Griewank (GR_{10})	10	Rastrigin (RA_{10})	10
Rosenbrock (R_{10})	10	Sum Squares (SS_{10})	10
Trid (T_{10})	10	**Zakharov** (Z_{10})	10
Griewank (GR_{20})	20	Rastrigin (RA_{20})	20
Rosenbrock (R_{20})	20	Sum Squares (SS_{20})	20
Zakharov (Z_{20})	20	Powell (PW_{24})	24
Dixon and Price (DP_{25})	25	Ackley (A_{30})	30
Levy (L_{30})	30	Sphere (SP_{30})	30

paper. Then, we compare the performance of the new C-GRASP with a genetic algorithm [14, 15], a scatter search algorithm [12], and a tabu search algorithm [9]. Lastly, we observe the performance of the new C-GRASP when implemented with sequential stopping rules of Hart [6].

3.1. **Test environment.** All experiments with C-GRASP were run on a Dell PowerEdge 2600 computer with dual 3.2 GHz 1 Mb cache XEON III processors and 6 Gb of memory running Red Hat Linux 3.2.3-53. C-GRASP was implemented in C++ and compiled with GNU `g++` version 3.2.3. The compiler options used were `-O6 -funroll-all-loops -fomit-frame-pointer -march=pentium4`. The algorithm used for random-number generation is an implementation of the Mersenne Twister algorithm described in [13]. For the experiments to follow, we made use of the test functions listed in Table 1. Note that the function definitions are given in the Appendix.

3.2. **C-GRASP comparison.** We first compare new C-GRASP with the one described in [10]. The two variants are compared on a set of 14 test functions taken from [10] and indicated in Table 1 in boldface. Since these test functions have known global minima, the two C-GRASP variants were run until the current objective function value $f(x)$ was significantly close to the global optimum $f(x^*)$. As in [7, 8, 9, 10, 18], we consider x and

TABLE 2. Original C-GRASP parameter values (from [10])

Parameter	Value	Parameter	Value
α	0.4	h (Starting value)	1
`MaxDirToTry`	30	`MaxIters`	200
`MaxNumIterNoImprov`	20	`NumTimesToRun`	20

TABLE 3. New C-GRASP parameter values (for results in Section 3.2 and Table 4)

Function	h_s	h_e	Function	h_s	h_e
BR	1	0.02	*EA*	1	0.1
GP	1	1	*SH*	1	0.01
$H_{3,4}$	0.5	0.05	$H_{6,4}$	0.5	0.005
R_2	1	0.01	R_5	1	0.01
R_{10}	1	0.01			
$S_{4,5}$	1	0.5	$S_{4,7}$	1	0.5
$S_{4,10}$	1	0.5			
Z_5	1	0.5	Z_{10}	1	0.005

x^* to be *significantly close* if

$$|f(x^*) - f(x)| \leq \varepsilon_1 |f(x^*)| + \varepsilon_2, \tag{1}$$

where $\varepsilon_1 = 10^{-4}$ and $\varepsilon_2 = 10^{-6}$.

For each of the 14 test functions, we ran each C-GRASP variant 100 times. Each run was independent of the others since different starting seeds were used for the pseudo random number generator. For both variants, the stopping criteria were finding a solution significantly close to the global optimum or completing 20 multi-start iterations. The parameters used in [10] are listed in Table 2. For the new C-GRASP, parameter ρ_{lo} was set to 0.7 for all of the test functions. The h_s and h_e values for each test function are listed in Table 3. The values were chosen to allow C-GRASP to make at least one call to the construction and local improvement procedures before 1,000 function evaluations were made.

We recorded the number of function evaluations and the elapsed time for the current solution to satisfy stopping criterion (1). The averages of these results are shown in Table 4 along with the percentage of times the algorithms found a significantly close solution. As is clear from Table 4, in almost all cases, there is a significant decrease in the number of function evaluations and running times needed to get significantly close to a global optimum, with no negative impact on the percentage of successful runs.

3.3. **Comparing C-GRASP with other heuristics.** We compare the new C-GRASP with three heuristics from the literature:

(1) Genetic Algorithm for Numerical Optimization of COnstrained Problems (*Genocop III*) [14, 15],

TABLE 4. Comparison between the new C-GRASP variant and the original C-GRASP from [10].

	Original C-GRASP			New C-GRASP		
Fn.	Fn. Evals	Time (s)	% Sig. Cl.	Fn. Evals	Time (s)	% Sig. Cl.
BR	59,857	0.0016	100	10,090	0.0011	100
EA	89,630	0.0042	100	5,093	0.0008	100
GP	29	0.0000	100	53	0.0000	100
SH	82,363	0.0078	100	18,608	0.0032	100
$H_{3,4}$	20,743	0.0026	100	1,719	0.0006	100
$H_{6,4}$	79,685	0.0140	100	29,894	0.0122	100
R_2	1,158,350	0.0132	100	23,544	0.0021	100
R_5	6,205,503	1.7520	100	182,520	0.0148	100
R_{10}	20,282,529	11.4388	99	725,281	0.2739	100
$S_{4,5}$	5,545,982	2.3316	100	9,274	0.0021	100
$S_{4,7}$	4,052,800	2.3768	100	11,766	0.0027	100
$S_{4,10}$	4,701,358	3.5172	100	17,612	0.0044	100
Z_5	959	0.0000	100	12,467	0.0025	100
Z_{10}	3,607,653	1.0346	100	2,297,937	1.1090	100

(2) Scatter Search (*SS*) [12], and
(3) Directed Tabu Search (DTS_{APS}) [9].

We make this comparison using all but two of the 42 multimodal test functions listed in Table 1, all with known global minima. We exclude functions R_5 and Z_5 since these function were not considered in the studies describing the three heuristics with which we compare C-GRASP.

In the C-GRASP local improvement procedure, we impose the restriction that the maximum number of points that can be evaluated in any neighborhood be limited to 1000, i.e. `MaxPointsToExamine` ≤ 1000.

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. As in [9, 12], we then say that the heuristic has solved the problem if

$$GAP \leq \begin{cases} \varepsilon & \text{if } f(x^*) = 0 \\ \varepsilon \cdot |f(x^*)| & \text{if } f(x^*) \neq 0, \end{cases} \tag{2}$$

where $\varepsilon = 0.001$.

At several points during each run, *GAP* values were computed. The average final *GAP* values over all 40 test functions are listed for each heuristic in Table 5. The results for C-GRASP are averages taken over all test functions and obtained by running the heuristic 100 times for each test function. DTS_{APS} was run 7 times for each test function. The data for the DTS_{APS} were taken from [9] while the data for *Genocop III* and *SS* are from [12]. Since *Genocop III* performed extremely poor for SC_6, two rows in Table 5 are devoted to this heuristic. Also listed in this table are results for C-GRASP if the function Z_{20} is not considered. Because C-GRASP encountered difficulties with function Z_{20}, its *GAP*

TABLE 5. Average *GAP* values (over all functions in Table 6).

Function evaluations	100	500	1,000	5,000	10,000	20,000	50,000
Genocop[1]	$5.37E+25$	$2.39E+17$	$1.13E+14$	636.37	399.52	320.84	313.34
Genocop[2]	1,335.45	611.30	379.03	335.81	328.66	324.72	321.20
Scatter Search	134.45	26.34	14.66	4.96	3.60	3.52	3.46
DTS_{APS}	50,400	43.06	24.26	4.22	1.80	1.70	1.29
C-GRASP[3]	23,610.61	10,185.84	1,341.70	6.20	4.73	3.92	3.02
C-GRASP[4]	24,208.75	10,442.53	1,371.62	1.87	0.37	0.07	0.03

[1] Average values over all test problems. [2] Average values over all test problems except problem SC_6.
[3] Average values over all test problems. [4] Average values over all test problems except problem Z_{20}.

values are higher than those of DTS_{APS}. However, when Z_{20} is not considered, C-GRASP does better than all the other heuristics from 5,000 function evaluations onward. To really compare the heuristics, one needs to look at the performance on a function by function basis. For C-GRASP, this performance is given in Table 6. At 50,000 function evaluations, all of the C-GRASP *GAP* values are less than 1 with the exception of Z_{20} which has a *GAP* value of 119.7624. Laguna and Marti [12] report that the final *GAP* values for *SS* are less than 1 for all but four functions: SC_6, RA_{10}, R_{20}, and A_{30} with *GAP* values of 118.4341, 9.9496, 2.2441, and 5.5033, respectively. However, [9, 12, 14, 15] do not report *GAP* results for individual test functions.

As an alternative to comparing actual *GAP* values, Figure 7 counts the number of test functions solved (according to Inequality (2)) by each algorithm as a function of the number of function evaluations. This figure shows C-GRASP performing competitively with the other algorithms, solving at least as many test functions from 10,000 function evaluations onward, and solving 33 of the 40 test problems when reaching 50,000 function evaluations.

3.4. **C-GRASP with Hart's sequential stopping rule.** Hart [6] considers a local search algorithm $L : S \to S$ that takes a sample on S and finds a sample that is a local minimum of f and defines the general multistart random search (*MSRS*) algorithm as follows:

Algorithm MSRS: Let $\xi_1, \xi_2, \ldots$ be independent and identically distributed random vectors with common distribution G on S. Let $(X_1, Y_1), (X_2, Y_2), \ldots$ be defined by

Step 1: $X_1 = L(\xi_1)$ and $Y_1 = f(X_1)$.
Step $k+1$: If $f(L(\xi_{k+1})) \le Y_k$, then $X_{k+1} = L(\xi_{k+1})$ and $Y_{k+1} = f(X_{k+1})$.
Else $X_{k+1} = X_k$ and $Y_{k+1} = Y_k$.

For $j = 2, \ldots, r$, where r is the number of steps taken in the *MSRS* algorithm, let

$$\tau_j(r) = \begin{cases} 0 & \text{if } |\{k \mid 1 \le k < \tau_{j-1}(r),\ Y_k \ne Y_{\tau_{j-1}(r)}\}| = 0 \\ \sup\{k \mid 1 \le k < \tau_{j-1}(r),\ Y_k \ne Y_{\tau_{j-1}(r)}\} & \text{otherwise,} \end{cases}$$

TABLE 6. Average *GAP* value for each test function for the new C-GRASP. Boldface entries denote *GAP* values satisfying Inequality (2). Results are grouped by function dimensions.

Fn.	h_s	h_e	100	500	1,000	5,000	10,000	20,000	50,000
BE	0.1	0.05	0.0036	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
B_2	1	0.1	0.4807	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
BO	0.1	0.05	2.5982	0.0032	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
BR	0.1	0.05	0.0084	0.0035	**0.0001**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
EA	1	0.1	0.0089	0.0089	0.0089	**0.0004**	**0.0004**	**0.0004**	**0.0004**
GP	0.1	0.05	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
M	0.1	0.05	**0.0004**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
R_2	1	0.1	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
SC_2	5	0.25	119.4013	0.3048	0.2368	0.1612	0.0103	0.0065	**0.0001**
SH	0.1	0.05	0.2498	**0.1617**	**0.1617**	**0.1511**	**0.1511**	**0.1511**	**0.0208**
CA	0.1	0.05	0.0405	**0.0002**	**0.0002**	**0.0002**	**0.0001**	**0.0001**	**0.0001**
Z_2	1	0.1	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
SP_3	0.1	0.05	0.5376	**0.0010**	**0.0010**	**0.0001**	**0.0001**	**0.0001**	**0.0001**
$H_{3,4}$	0.1	0.05	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
CV	1	0.05	11.3623	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
$P_{4,\frac{1}{2}}$	0.1	0.0125	7.2999	0.8879	0.1328	0.0359	0.0118	0.0031	0.0031
$P^0_{4,10}$	0.1	0.05	2463.2448	0.6782	0.0109	0.0012	**0.0004**	**0.0003**	**0.0000**
$PS_{4,b}$ [1]	0.1	0.05	0.8433	0.0069	0.0033	**0.0007**	**0.0005**	**0.0002**	**0.0000**
$S_{4,5}$	0.1	0.05	9.2904	6.9209	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
$S_{4,7}$	0.1	0.05	9.5018	5.9017	**0.0001**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
$S_{4,10}$	0.1	0.05	9.5240	7.2325	0.8944	**0.0001**	**0.0001**	**0.0001**	**0.0001**
$H_{6,4}$	0.1	0.05	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
SC_6	50	0.25	835.8956	221.9143	144.6920	1.6621	1.2104	0.1214	0.0233
T_6	1	0.1	335.0119	192.9117	15.3233	**0.0000**	**0.0000**	**0.0000**	**0.0000**
GR_{10}	10	0.25	20.3141	20.3141	12.4181	**0.0000**	**0.0000**	**0.0000**	**0.0000**
RA_{10}	2	0.1	77.2000	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
R_{10}	2	0.05	8064.0275	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
SS_{10}	1	0.1	259.9448	36.7055	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
T_{10}	20	0.1	6402.6928	410.7253	64.8220	18.6801	6.6901	1.5025	0.2181
Z_{10}	1	0.1	37.1243	37.1243	27.9358	0.2051	0.0646	0.0059	0.0011
GR_{20}	10	0.25	61.0211	61.0211	61.0211	31.7019	4.4302	**0.0000**	**0.0000**
RA_{20}	2	0.1	258.8922	188.1984	102.4020	**0.0000**	**0.0000**	**0.0000**	**0.0000**
R_{20}	2	0.1	210591.5859	89313.8553	15140.9928	**0.0000**	**0.0000**	**0.0000**	**0.0000**
SS_{20}	1	0.1	1934.5603	1211.4429	732.6502	**0.0000**	**0.0000**	**0.0000**	**0.0000**
Z_{20}	2	0.05	283.0573	174.8449	174.8449	174.8449	174.8449	153.9820	119.7624
PW_{24}	2	0.1	5296.1979	1056.8398	434.9449	1.4250	0.1948	0.1528	0.0806
DP_{25}	5	0.2	707004.8655	314195.4226	36508.5904	0.6680	0.6680	0.6680	0.6680
A_{30}	5	0.05	19.3023	18.6985	17.5581	0.6078	**0.0000**	**0.0000**	**0.0000**
L_{30}	2	0.05	126.3283	111.4947	96.5772	4.1265	0.8559	0.0088	**0.0000**
SP_{30}	1	0.05	181.8665	159.9949	131.7025	13.5988	**0.0000**	**0.0000**	**0.0000**

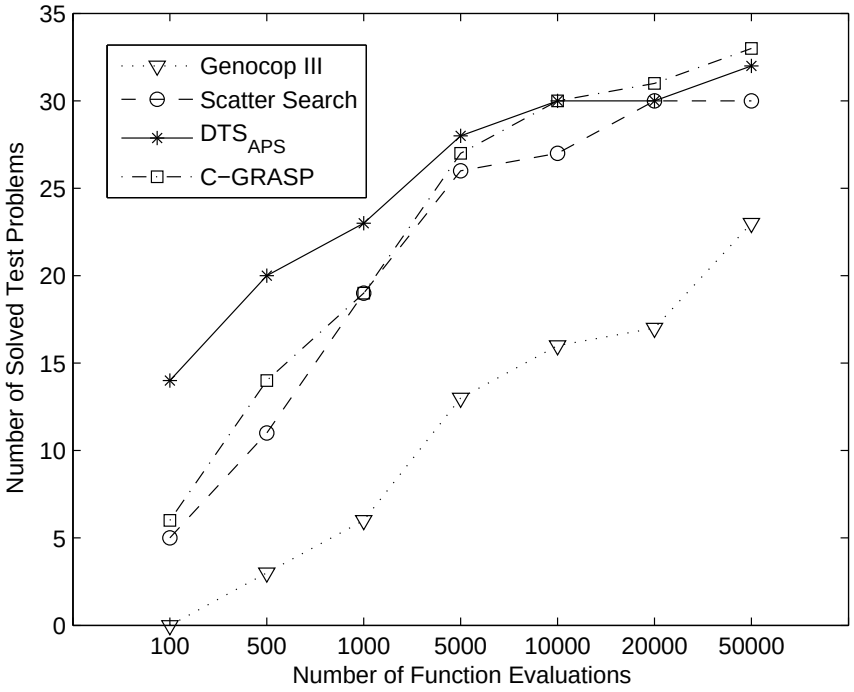

FIGURE 7. Number of problems solved as a function of number of function evaluations.

with $\tau_1(r) = r$, and let

$$\rho_r(\varepsilon) = \sup\{k \mid \tau_k(r) > 0, Y_{\tau_k(r)} \leq Y_r + \varepsilon\},$$

and

$$\Gamma_r(\varepsilon) = \begin{cases} |\{Y_i \mid Y_i \leq Y_r + \varepsilon,\ i = \tau_2(r)+1, \ldots, r-1\}| & \text{if } \tau_2(r)+1 < r \\ 0 & \text{otherwise.} \end{cases}$$

Let $\hat{\rho}_r(\varepsilon) = \rho_r(\varepsilon) + \Gamma_r(\varepsilon)$. The term $\hat{\rho}_r(\varepsilon)/r$ is used to estimate $\rho_\varepsilon = \Pr(f(x) \leq f(x^*) + \varepsilon)$. Hart [6] shows that $\hat{\rho}_r(\varepsilon)/r$ is a better estimate to ρ_ε than $\rho_r(\varepsilon)/r$, the estimate defined by Dorea [1]. It can be easily seen from the above definitions that $\Gamma_r(\varepsilon) = r - \tau_2(r) - 1$, i.e. $\Gamma_r(\varepsilon)$ counts the number of steps whose Y value equals the value Y_r of the r-th step of *MSRS*.

Hart's stopping rule can be stated as follows: For a given $\varepsilon > 0$, $\delta > 0$, and $\beta \in (0,1)$, terminate *Algorithm MSRS* if $r \geq 2$ and

$$\Phi(2\delta\sqrt{r}) - \Phi(-2\delta\sqrt{r}) - (1 - \hat{\rho}_r(\varepsilon)/r)^r \geq 1 - \beta,$$

where Φ is the cumulative distribution function of the standard normal, i.e

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-y^2/2} dy.$$

TABLE 7. Average number of multi-starts and average *GAP* value for each test function for the new C-GRASP implemented with Hart's stopping rule [6]. Boldface items denote those *GAP* values satisfying Inequality (2).

Function	Avg, # MS	*GAP*	Function	Avg. # MS	*GAP*
BE	8.4	**0.0000**	B_2	8	**0.0000**
BO	8	**0.0000**	BR	8	**0.0000**
EA	8.4	**0.0000**	GP	8.2	**0.0000**
M	8	**0.0000**	R_2	8	**0.0000**
SC_2	8.4	**0.0000**	SH	8	**0.0003**
CA	8.3	**0.0000**	Z_2	8	**0.0000**
SP_3	8.2	**0.0000**	$H_{3,4}$	8.4	**0.0000**
CV	8.4	**0.0000**	$P_{4,\frac{1}{2}}$	9.3	0.0023
$P^0_{4,10}$	8.3	**0.0000**	${PS_{4,b}}^2$	8.4	**0.0000**
$S_{4,5}$	8.5	**0.0000**	$S_{4,7}$	8.3	**0.0000**
$S_{4,10}$	8.8	**0.0001**	$H_{6,4}$	8.5	**0.0000**
SC_6	8.1	**0.0000**	T_6	8	**0.0000**
GR_{10}	9.5	**0.0000**	RA_{10}	8	**0.0000**
R_{10}	8.6	**0.0000**	SS_{10}	8	**0.0000**
T_{10}	8	0.0029	Z_{10}	8.2	**0.0000**
GR_{20}	10.1	**0.0000**	RA_{20}	8	**0.0000**
R_{20}	8	**0.0000**	SS_{20}	8	**0.0000**
Z_{20}	9	0.0049	PW_{24}	8.3	**0.0010**
DP_{25}	8.2	**0.0000**	A_{30}	8	**0.0000**
L_{30}	8	**0.0000**	SP_{30}	8	**0.0000**

In this stopping rule, ε is the desired accuracy of the best solution found, δ is a limit on the difference between $\hat{\rho}_r(\varepsilon)/r$ and ρ_ε (ensures that r is sufficiently large for ρ_ε to be reliably estimated), and $1-\beta$ is the required probability of success.

We implemented Hart's stopping rules in C-GRASP and ran each test function 10 times. The three stopping-rule parameters were defined to be $\varepsilon = 0.001$, $\delta = 0.4$, and $\beta = 0.025$, and each function used the same h_s and h_e values as listed in Table 6. For each test function, we took the average *GAP* value when the algorithm finished. Table 7 presents these results for each function. Each function has associated with it two numbers: the first is the average number of multi-starts required for each run, and the second is the average final *GAP* value over the runs. From this table, one can see that when C-GRASP is run as a true multi-start algorithm using Hart's sequential stopping rule, almost all of the functions have a final *GAP* value satisfying Inequality (2). Only the runs on T_{10}, $P_{4,\frac{1}{2}}$, and Z_{20} did not satisfy the inequality, with final *GAP* values of 0.0029, 0.0023, and 0.0049 respectively. These *GAP* values are quite small and, all in all, C-GRASP did well on these test functions.

4. Concluding remarks

In this paper, we described several improvements to the original Continuous GRASP (C-GRASP) metaheuristic of Hirsch et al. [10]. As seen in Section 2, the metaheuristic is still easy to describe and implement. Furthermore, the improvements do not detract from C-GRASP being generally applicable to global optimization problems. Section 3 begins by validating the improvements. In addition, the new C-GRASP was compared with three other metaheuristics from the recent literature, with encouraging results. The approach was shown to converge when Hart's sequential stopping rule [6] was implemented. These results demonstrate the potential of C-GRASP. Since C-GRASP makes no use of derivative nor *a priori* information, it is a well-suited approach for solving general global optimization problems.

Acknowledgment

The research of the second author was partially supported by NSF, NIH, and CRDF grants. The authors thank Abdel-Rahman Hedar and Masao Fukushima for the data of DTS_{APS}, *SS*, and *Genocop III* used in Figure 7.

References

[1] C. C. Y. Dorea. Stopping rules for a random optimization method. *SIAM Journal on Control and Optimization*, 28(4):841–850, 1990.

[2] 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.

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

[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. In G. Goods and J. Hartmanis, editors, *Lecture Notes in Computer Science*, volume 455. Springer Verlag, Berlin, 1990.

[6] W. E. Hart. Sequential stopping rules for random optimization methods with application to multistart local search. *SIAM Journal of Optimization*, 9(1):270–290, 1998.

[7] A. R. Hedar and M. Fukushima. Hybrid simulated annealing and direct search method for nonlinear unconstrained global optimization. *Optimization Methods and Software*, 17:891–912, 2002.

[8] A. R. Hedar and M. Fukushima. Minimizing multimodal functions by simplex coding genetic algorithms. *Optimization Methods and Software*, 18:265–282, 2003.

[9] A. R. Hedar and M. Fukushima. Tabu search directed by direct search methods for nonlinear global optimization. *European Journal of Operational Research*, 170:329–349, 2006.

[10] M. J. Hirsch, C. N. Meneses, P. M. Pardalos, and M. G. C. Resende. Global optimization by continuous GRASP. *Optimization Letters (to appear)*, 2006.

[11] M. J. Hirsch, P. M. Pardalos, and M. G. C. Resende. Sensor registration in a sensor network by continuous GRASP. *Proceedings of the Military Communications Conference (to appear)*, 2006.

[12] M. Laguna and R. Martí. Experimental testing of advanced Scatter Search designs for global optimization of multimodal functions. *Journal of Global Optimization*, 33:235–255, 2005.
[13] 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.
[14] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer Verlag, Berlin, 2nd edition, 2004.
[15] Z. Michalewicz and G. Nazhiyath. Genocop III: A Co-evolutionary Algorithm for Numerical Optimization Problems with Nonlinear Constraints. In *Proceedings of the 2nd IEEE ICEC*, Perth, Australia, 1995.
[16] P. M. Pardalos and M. G. C. Resende, editors. *Handbook of Applied Optimization*. Oxford University Press, New York, 2002.
[17] 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, 2003.
[18] P. Siarry, G. Berthiau, F. Durbin, and J. Haussy. Enhanced simulated annealing for globally minimizing functions of many continuous variables. *ACM Transactions on Mathematical Software*, 23(2):209–228, 1997.

APPENDIX A. FUNCTION DEFINITIONS

This appendix lists all the functions used in the experiments described in this paper.

(A_n) **Ackley Function**
Definition: $A_n(x) = -20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)} + 20 + e$
Domain: $[-15,30]^n$
Global Minimum: $x^* = (0,\ldots,0)$; $A_n(x^*) = 0$

(BE) **Beale Function**
Definition: $BE(x) = (1.5 - x_1 - x_1x_2)^2 + (2.25 - x_1 - x_1x_2^2)^2 + (2.625 - x_1 - x_1x_2^3)^2$
Domain: $[-4.5,4.5]^2$
Global Minimum: $x^* = (3,\frac{1}{2})$; $BE(x^*) = 0$

(B_2) **Bohachevsky Function**
Definition: $B_2(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7$
Domain: $[-50,100]^2$
Global Minimum: $x^* = (0,0)$; $B_2(x^*) = 0$

(BO) **Booth Function**
Definition: $BO(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$
Domain: $[-10,10]^2$
Global Minimum: $x^* = (1,3)$; $BO(x^*) = 0$

(*BR*) **Branin Function**
Definition: $BR(x) = (x_2 - \frac{5}{4\pi^2}x_1^2 + \frac{1}{\pi}5x_1 - 6)^2 + 10(1 - \frac{1}{8\pi})\cos(x_1) + 10$
Domain: $[-5, 15]^2$
Global Minimum (one of several): $x^* = (\pi, 2.275)$; $BR(x^*) = 0.397887$

(*CV*) **Colville Function** (also called **Wood Function**)
Definition: $CV(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2$
$+10.1\left[(x_2 - 1)^2 + (x_4 - 1)^2\right] + 19.8(x_2 - 1)(x_4 - 1)$
Domain: $[-10, 10]^4$
Global Minimum: $x^* = (1, 1, 1, 1)$; $CV(x^*) = 0$

(DP_n) **Dixon and Price Function**
Definition: $DP_n(x) = (x_1 - 1)^2 + \sum_{i=2}^{n} i(2x_i^2 - x_{i-1})^2$
Domain: $[-10, 10]^n$
Global Minimum: $x_i^* = 2^{-\frac{2^i - 2}{2^i}}$, $\forall i = 1, \ldots, n$; $DP_n(x^*) = 0$

(*EA*) **Easom Function**
Definition: $EA(x) = -\cos(x_1)\cos(x_2)e^{-(x_1-\pi)^2-(x_2-\pi)^2}$
Domain: $[-100, 100]^2$
Global Minimum: $x^* = (\pi, \pi)$; $EA(x^*) = -1$

(*GP*) **Goldstein and Price Function**
Definition: $GP(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$
$[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$
Domain: $[-2, 2]^2$
Global Minimum: $x^* = (0, -1)$; $GP(x^*) = 3$

(GR_n) **Griewank Function**
Definition: $GR_n(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$
Domain: $[-300, 600]^n$
Global Minimum: $x^* = (0, \ldots, 0)$; $GR_n(x^*) = 0$

($H_{n,m}$) **Hartmann Function**
Definition: $H_{n,m}(x) = -\sum_{i=1}^{m} \alpha_i e^{-\sum_{j=1}^{n} A_{ij}^{(n)}(x_j - P_{ij}^{(n)})^2}$
Domain: $[0, 1]^n$
Global Minimum: $(n = 3,\ m = 4)$ $x^* = (0.114614, 0.555649, 0.852547)$; $H_{3,4}(x^*) = -3.86278$
Global Minimum: $(n = 6,\ m = 4)$ $x^* = (0.201690, 0.150011, 0.476874, 0.275332, 0.311652, 0.657300)$;
$H_{6,4}(x^*) = -3.32237$
Parameters:

$$A^{(3)} = \begin{bmatrix} 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \end{bmatrix}$$

$$P^{(3)} = 10^{-4} \begin{bmatrix} 6890 & 1170 & 2673 \\ 4699 & 4387 & 7470 \\ 1091 & 8732 & 5547 \\ 381 & 5743 & 8838 \end{bmatrix}$$

$$A^{(6)} = \begin{bmatrix} 10 & 3 & 17 & 3.05 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix}$$

$$P^{(6)} = 10^{-4} \begin{bmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{bmatrix}$$

$\alpha = [1, 1.2, 3, 3.2]$

(L_n) Levy Function
Definition: $L_n(x) = \sin^2(\pi y_1) + \sum_{i=1}^{n-1} \left[(y_i - 1)^2 (1 + 10\sin^2(\pi y_i + 1))\right] + (y_n - 1)^2 (1 + 10\sin^2(2\pi y_n))$
Domain: $[-10, 10]^n$
Global Minimum: $x^* = (1, \ldots, 1)$; $L_n(x^*) = 0$
Parameters: $y_i = 1 + \frac{x_i - 1}{4}$. $\forall\, i = 1, \ldots, n$

(M) Matyas Function
Definition: $M(x) = 0.26(x_1^2 + x_2^2) - 0.48 x_1 x_2$
Domain: $[-5, 10]^2$
Global Minimum: $x^* = (0, 0)$; $M(x^*) = 0$

($P_{n,\beta}$) Perm Function
Definition: $P_{n,\beta}(x) = \sum_{k=1}^{n} \left[\sum_{i=1}^{n} (i^k + \beta)((\frac{x_i}{i})^k - 1)\right]^2$
Domain: $[-n, n]^n$
Global Minimum: $x^* = (1, 2, \ldots, n)$; $P_{n,\beta}(x^*) = 0$

($P^0_{n,\beta}$) Perm$_0$ Function
Definition: $P^0_{n,\beta}(x) = \sum_{k=1}^{n} [\sum_{i=1}^{n} (i + \beta)((x_i^k - (\frac{1}{i})^k))]^2$
Domain: $[-n, n]^n$
Global Minimum: $x^* = (1, \frac{1}{2}, \ldots, \frac{1}{n})$; $P^0_{n,\beta}(x^*) = 0$

(PW_n) Powell Function
Definition: $PW_n(x) = \sum_{i=1}^{\frac{n}{4}} \Big[(x_{4i-3} + 10x_{4i-2})^2 + 5(x_{4i-1} - x_{4i})^2$
$+ (x_{4i-2} - 2x_{4i-1})^4 + 10(x_{4i-3} - x_{4i})^4\Big]$
Domain: $[-4, 5]^n$
Global Minimum: $x^* = (3, -1, 0, 1, 3, \ldots, 3, -1, 0, 1)$; $PW_n(x^*) = 0$

($PS_{n,b}$) Power Sum Function
Definition: $PS_{n,b}(x) = \sum_{k=1}^{n} ((\sum_{i=1}^{n} x_i^k) - b_k)^2$
Domain: $[0, n]^n$
Global Minimum ($n = 4, b = \{8, 18, 44, 114\}$): $x^* = (1, 2, 2, 3)$; $PS_{4,\{8,18,44,114\}}(x^*) = 0$

(RA_n) Rastrigin Function
Definition: $RA_n(x) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i))$
Domain: $[-2.56, 5.12]^n$

Global Minimum: $x^* = (0,\ldots,0)$; $RA_n(x^*) = 0$

(R_n) **Rosenbrock Function**
Definition: $R_n(x) = \sum_{j=1}^{n-1}\left[100(x_j^2 - x_{j+1})^2 + (x_j - 1)^2\right]$
Domain: $[-10,10]^n$
Global Minimum: $x^* = (1,\ldots,1)$; $R_n(x^*) = 0$

(SC_n) **Schwefel Function**
Definition: $SC_n(x) = 418.9829n - \sum_{i=1}^{n} x_i \sin(\sqrt{|x_i|})$
Domain: $[-500,500]^n$
Global Minimum: $x^* = (420.9687,\ldots,420.9687)$; $SC_n(x^*) = 0$

($S_{4,m}$) **Shekel Function**
Function Definition: $S_{4,m}(x) = -\sum_{i=1}^{m}[(x-a_i)^T(x-a_i) + c_i]^{-1}$
Domain: $[0,10]^4$
Global Minimum: $x^* = (4,4,4,4)$; $S_{4,5}(x^*) = -10.15319538$, $S_{4,7}(x^*) = -10.40281868$, and $S_{4,10}(x^*) = -10.53628349$
Parameters:

$$a = \begin{bmatrix} 4.0 & 4.0 & 4.0 & 4.0 \\ 1.0 & 1.0 & 1.0 & 1.0 \\ 8.0 & 8.0 & 8.0 & 8.0 \\ 6.0 & 6.0 & 6.0 & 6.0 \\ 7.0 & 3.0 & 7.0 & 3.0 \\ 2.0 & 9.0 & 2.0 & 9.0 \\ 5.0 & 5.0 & 3.0 & 3.0 \\ 8.0 & 1.0 & 8.0 & 1.0 \\ 6.0 & 2.0 & 6.0 & 2.0 \\ 7.0 & 2.6 & 7.0 & 3.6 \end{bmatrix}$$

$c = [0.1, 0.2, 0.2, 0.4, 0.4, 0.6, 0.3, 0.7, 0.5, 0.5]$

(SH) **Shubert Function**
Definition: $SH(x) = \left[\sum_{i=1}^{5} i\cos[(i+1)x_1 + i]\right]\left[\sum_{i=1}^{5} i\cos[(i+1)x_2 + i]\right]$
Domain: $[-10,10]^2$
Global Minimum (one of several): $x^* = (5.48242188, 4.85742188)$; $SH(x^*) = -186.7309$

(CA) **Six-Hump CamelBack Function**
Definition: $CA(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$
Domain: $[-5,5]^2$
Global Minimum (one of several): $x^* = (0.08984375, -0.71289062)$; $CA(x^*) = -1.03162801$

(SP_n) **Sphere Function**
Definition: $SP_n(x) = \sum_{i=1}^{n} x_i^2$
Domain: $[-2.56, 5.12]^n$
Global Minimum: $x^* = (0,\ldots,0)$; $SP_n(x^*) = 0$
N.B.: The De Joung Function (DJ) is just a special case of the Sphere Function, i.e $DJ(x) = SP_3(x)$

(SS_n) **Sum of Squares Function**
Definition: $SS_n(x) = \sum_{i=1}^{n} ix_i^2$
Domain: $[-5,10]^n$
Global Minimum: $x^* = (0,\ldots,0)$; $f(x^*) = 0$

(T_n) **Trid Function**
Definition: $T_n(x) = \sum_{i=1}^{n}(x_i-1)^2 - \sum_{i=2}^{n} x_i x_{i-1}$
Domain: $[-n^2,n^2]^n$
Global Minimum: $x_i^* = i(n+1-i)$, $\forall\, i = 1,\ldots,n$; $T_6(x^*) = -50$ and $T_{10}(x^*) = -210$

(Z_n) **Zakharov Function**
Definition: $Z_n(x) = \sum_{i=1}^{n} x_i^2 + (\sum_{i=1}^{n} 0.5ix_i)^2 + (\sum_{i=1}^{n} 0.5ix_i)^4$
Domain: $[-5,10]^n$
Global Minimum: $x^* = (0,\ldots,0)$; $Z_n(x^*) = 0$

(Michael J. Hirsch) RAYTHEON, INC., NETWORK CENTRIC SYSTEMS, P.O. BOX 12248, ST. PETERSBURG, FL, 33733-2248, AND DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING, UNIVERSITY OF FLORIDA, 303 WEIL HALL, GAINESVILLE, FL, 32611, USA.
E-mail address: mjh8787@ufl.edu

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

(Mauricio G. C. Resende) ALGORITHMS AND OPTIMIZATION RESEARCH DEPARTMENT, AT&T LABS RESEARCH, 180 PARK AVENUE, ROOM C241, FLORHAM PARK, NJ 07932 USA.
E-mail address: mgcr@research.att.com