J Glob Optim (2010) 48:289–310
DOI 10.1007/s10898-009-9494-z

Continuous GRASP with a local active-set method for bound-constrained global optimization

Ernesto G. Birgin · Erico M. Gozzi · Mauricio G. C. Resende · Ricardo M. A. Silva

Received: 27 April 2009 / Accepted: 9 November 2009 / Published online: 29 November 2009

Abstract Global optimization seeks a minimum or maximum of a multimodal function over a discrete or continuous domain. In this paper, we propose a hybrid heuristic—based on the CGRASP and GENCAN methods—for finding approximate solutions for continuous global optimization problems subject to box constraints. Experimental results illustrate the relative effectiveness of CGRASP–GENCAN on a set of benchmark multimodal test functions.

Keywords Global optimization · Stochastic methods · Active-set methods · Heuristic · CGRASP · GENCAN

1 Introduction

Global optimization [17] seeks a minimum or maximum of a multimodal function over a discrete or continuous domain. In its minimization form, global optimization is stated

This research was done while Ricardo M. A. Silva was a visiting post-doctoral scholar at AT&T Labs Research. His work was partially funded by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil.

E. G. Birgin · E. M. Gozzi
Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, Brazil

E. G. Birgin
e-mail: egbirgin@ime.usp.br

E. M. Gozzi
e-mail: erico.gozzi@ime.usp.br

M. G. C. Resende (✉)
Algorithms and Optimization Research Department, AT&T Labs Research, Florham Park, NJ, USA
e-mail: mgcr@research.att.com

R. M. A. Silva
Department of Computer Science, Federal University of Lavras, Lavras, Brazil
e-mail: rmas@dcc.ufla.br

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*.

The problem of minimizing a continuous function with bounds on the variables has many practical applications [14]. Moreover, box-constrained minimization algorithms are used as subroutines for solving the subproblems that appear in many augmented Lagrangian and penalty methods for general constrained optimization (see for example, [3,4,10,18,23]). As mentioned in [3] and rigorously proved in [10], one of the advantages of the augmented Lagrangian approach for solving nonlinear programming problems is its intrinsic adaptability to the global optimization problem. Namely, if one knows how to globally solve simple subproblems, the augmented Lagrangian method allows one to globally solve the original constrained optimization problem. In this sense, developing efficient methods for global bound-constrained minimization is a step toward the development of efficient methods for general constrained global optimization.

Among several methods proposed for continuous global optimization problems subject to box constraints is the Continuous GRASP (CGRASP) of Hirsch et al. [15,16]. CGRASP is an adaptation for solving continuous global optimization problems of the greedy randomized adaptive search procedure (GRASP) of Feo and Resende [12,13]. Like a GRASP, a CGRASP is a multi-start procedure where a starting solution for local improvement is constructed in a greedy randomized fashion. The main difference is that an iteration of CGRASP does not consist of a single greedy randomized construction followed by local improvement, but rather a series of construction-local improvement cycles where, like in a GRASP, the output of the construction serves as input to local improvement, but unlike GRASP, the output of local improvement serves as input to the construction. Periodically, the algorithm is restarted from a randomly generated starting solution.

The local improvement procedures in the CGRASP heuristics introduced in [15,16] sample points around the solution produced by the global greedy randomized procedure. Since they only make function evaluations and do not use gradient information, they can be used for local optimization of any type of function, including ones that are not smooth. In this paper, we adapt CGRASP for global optimization of functions for which gradients can be computed. To to this, we use GENCAN [6], an active-set method for bound-constrained local minimization.

GENCAN adopts the leaving-face criterion of Birgin and Martinez [5] that employs spectral projected gradients defined in [8,9]. For the internal-to-the-face minimization, GENCAN uses a general algorithm with a line search that combines backtracking and extrapolation. In the present available implementation [7], each step of GENCAN computes the direction inside the face using a truncated-Newton approach with incremental quotients to approximate the matrix-vector products and memoryless BFGS preconditioners.

CGRASP–GENCAN is a hybrid algorithm based on the CGRASP and GENCAN methods for finding approximate solutions for continuous global optimization problems subject to box constraints. CGRASP–GENCAN consists in substituting the point-sampling local improvement procedure in CGRASP by the GENCAN method. GENCAN requires smoothness of the objective function being optimized and uses its gradient in the optimization process. We aim to show that, on one hand, using GENCAN in the local-improvement phase of CGRASP enables the method to find highly accurate solutions that satisfy local first-order optimality conditions. On the other hand, we also aim to show that, when using the same GENCAN strategy for the local-optimization phase, the CGRASP globalization strategy outperforms classical global optimization techniques, like Random Linkage [21,22] and Tunneling [2,19,20].

Experimental results illustrate the relative effectiveness of CGRASP–GENCAN on a set of benchmark multimodal test functions.

This paper is organized as follows. The CGRASP–GENCAN algorithm is described in Sect. 2. Section 3 reports on computational results comparing CGRASP–GENCAN with five other global minimization algorithms that make use of GENCAN for local minimization. Concluding remarks are made in Sect. 4. Appendix A lists the 41 multimodal test functions used in the computational experiments.

2 Continuous GRASP with GENCAN

Pseudo-code of the CGRASP–GENCAN method for global minimization is shown in Fig. 1. The procedure takes as input the problem dimension n, lower and upper bound vectors $\ell \in \mathbb{R}^n$ and $u \in \mathbb{R}^n$, respectively (such that $\ell_i < u_i$, for $i = 1, \ldots, n$), the objective function $f(\cdot)$ and its gradient $g(\cdot)$, as well as the parameters h_s and h_e, used to define the starting and ending grid discretization densities, respectively.

Line 1 of the pseudo-code initializes to infinity the objective function value f^* of the best solution found. Since CGRASP–GENCAN 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 or the elapsed time. Since different implementations of CGRASP–GENCAN can have different stopping criteria, we list line 2 in general form.

As long as the stopping criteria of line 2 are not satisfied, another iteration takes place, as seen in lines 3–17. In line 3 the initial solution x is set to a random point distributed uniformly in the box $\{x \in \mathbb{R}^n \mid \ell_i \le x_i \le u_i : i = 1, \ldots, n\}$. Parameter h, that controls the discretization density of the search space, is re-initialized to h_s in line 4. The inner loop in lines 5 to 17 is repeated while the discretization density parameter $h \ge h_e$, for some h_e such that $0 < h_e < h_s$. Variable $\mathtt{Impr_C}$ ($\mathtt{Impr_L}$) is true if and only if the randomized

```
procedure CGRASP-GENCAN(n, ℓ, u, f(·), g(·), h_s, h_e)
1       f* ← ∞;
2       while stopping criteria not satisfied 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] ← GENCAN(x, f(·), g(·), n, ℓ, u, 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 CGRASP-GENCAN;
```

Fig. 1 Pseudo-code of the CGRASP–GENCAN algorithm greedy construction (GENCAN) phase improves upon its starting solution. They are set to `false` in lines 6 and 7. The construction and GENCAN phases are then called sequentially in lines 8 and 9, respectively. The solution x returned from construction phase in line 8 is input to GENCAN in line 9. The solution returned from GENCAN is compared against the current best solution in line 10. If necessary, the current best solution is updated with the returned solution in lines 11 and 12. In lines 14–16, if variables $\texttt{Impr}_\texttt{C}$ and $\texttt{Impr}_\texttt{L}$ remain `false`, then since neither the construction nor GENCAN improved upon x, the grid density is increased by halving h. When the stopping criteria are satisfied, x^*, the best solution found, is returned in line 19.

The randomized greedy construction procedure of Hirsch et al. [15] is shown in Fig. 2. The input is a solution vector x. To start, the algorithm allows all coordinates of x to change (i.e. they are unfixed). 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

```
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_1, x_2, ..., x_{i−1}, z_i, x_{i+1}, ..., x_n);
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;
```

Fig. 2 Pseudo-code of the CGRASP–GENCAN randomized greedy construction procedure of x held at their current values. In lines 10 and 11, the value z_i for the ith coordinate that minimizes the objective function in the line search, together with the objective function value G_i, are saved.

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 minimum G_i values over all unfixed coordinates of x, and $\alpha \in [0, 1]$ is a parameter chosen at random in line 2. In line 24, a coordinate (say coordinate j) is randomly selected from the `RCL`. In line 25, the procedure checks whether x_j and z_j are equal. If they are, `ReUse` is set to `true` in line 26. Otherwise, in lines 28–30, `ReUse` is set to `false`, $\mathtt{Impr}_\mathtt{C}$ is set to `true`, and x_j is set equal to z_j. Finally, in line 32, the coordinate j of x is fixed 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 $\mathtt{Impr}_\mathtt{C}$ are returned.

As discussed in [15], the `ReUse` parameter in the construction procedure avoids unnecessary line searches and function evaluations. If we reach line 25 and determine that $x_j = z_j$, then in the next iteration of the loop in lines 4 to 33, all z_i, for $i \in$ `UnFixed`, would remain unchanged, and therefore the line searches in line 10 and function evaluations in line 11 can be avoided.

3 Experimental results

We study the performance of the CGRASP–GENCAN heuristic and five other algorithms that also use GENCAN as their local improvement component. These algorithms either use Random Linkage [21,22] or Tunneling with Lissajous curves [2,19,20] as their global component. For the experiments to follow, we make use of the 52 test functions used in [1,21,22]. The test functions are listed in Appendix A.

3.1 Test environment

All experiments were carried out on an AMD K8 1.8 GHz processor with 1 Gb of RAM running Ubuntu Linux version 6.06. CGRASP–GENCAN was implemented in Fortran and compiled with the GNU Fortran (`g77`) compiler, version 3.4.6, using compiler option `-O4`.

3.2 Comparing the methods

We analyze the effectiveness and robustness of CGRASP–GENCAN and other well-known global optimization methods. As aforementioned, all methods differ in the global phase and coincide in the local phase where all use GENCAN as a local bound-constrained minimization solver.

The four random linkage heuristics as well as the tunneling heuristic were run using the default parameters recommended in [21] and [2], respectively. Each of the six algorithms was limited to at most 20,000 calls to GENCAN. If the algorithm was able to find the global minimum, it was stopped and run-related statistics were collected, including the CPU time and the number of calls to GENCAN.

Tables 1, 2, 3, 4, 5, and 6 summarize the performances of the six heuristics. All heuristics used GENCAN as the local minimization component. For their global optimization components, four used random linkage (with $\sigma = 0, 1, 2, 4$), one used tunneling with

Table 1 Experimental results for *Multistart* (*Random Linkage* with $\sigma = 0$) with GENCAN

Prob	n	Multistart—random linkage ($\sigma = 0$)					
		$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$
1	2	−1.14265	1	0.00	9	10	1
2	2	−377.60873	1	0.00	8	5	1
3	2	−186.73091	31	0.01	244	209	31
4a	2	−186.73091	143	0.06	1,486	1,104	143
4b	2	−186.73091	125	0.06	1,348	996	125
5	5	0.00000	27	0.02	566	419	27
5	8	0.00000	27	0.03	644	471	27
5	10	0.00000	4	0.01	148	106	4
5	20	0.00000	50	0.30	3,483	2,098	50
5	30	0.00000	6	0.16	1,190	701	6
6	**4**	**−21.39248**	**214**	**0.28**	**7,185**	**5,438**	**214**
7	2	−24.06250	74	0.02	529	493	74
8	7	−274.16316	3	0.00	33	29	3
9	2	−176.541793	1	0.00	8	7	1
10	2	−3.30687	1,426	0.57	14,797	11,908	1,426
11	3	−0.49026	2	0.00	21	17	2
12	2	0.00740	678	0.23	7,456	4,776	678
13	2	124.36218	1,338	61.99	1,024,276	1,025,251	1,338
14	4	0.00031	1,223	24.46	503,301	484,418	1,223
15	4	85,822.20160	1	0.00	11	12	1
16	5	0.00000	26	0.01	285	219	26
17	2	0.00000	1	0.00	6	7	1
18	2	0.00000	1	0.00	5	6	1
19a	4	−10.15320	3	0.00	40	26	3
19b	4	−10.53641	3	0.00	51	29	3
20	4	0.00000	1	0.00	14	15	1
21	2	−1.03163	2	0.00	19	16	2
22	2	−186.73091	31	0.01	244	209	31
23	2	−78.33233	3	0.00	33	31	3
23	3	−117.49850	3	0.00	35	33	3
23	4	−156.66466	40	0.02	404	417	40
24	2	0.00000	1	0.00	6	6	1
25	2	−0.40746	1	0.00	13	14	1
26	2	−18.05870	1	0.00	9	10	1
27	2	−227.76575	1	0.00	14	15	1
28	2	−2,429.41477	1	0.00	11	12	1
29	2	−2.00000	133	0.04	1,272	982	133
30	2	0.39789	1	0.00	11	8	1
31	1	−3.37290	5	0.00	38	30	5
32	2	1.00000	18	0.02	310	227	18
33	1	7.00000	1	0.00	3	3	1

Table 1 continued

Prob	n	Multistart—random linkage ($\sigma = 0$)					
		$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$
34	4	0.00000	1	0.00	30	21	1
35	2	0.00000	2	0.00	24	21	2
36	4	0.00000	1	0.00	41	27	1
37	10	0.00000	1	0.00	3	4	1
37	20	0.00000	1	0.00	7	8	1
37	30	0.00000	1,009	1.42	4,778	5,787	1,009
37	40	0.00000	148	0.26	651	799	148
38	10	0.00000	1	0.00	2	3	1
39	10	2.50000	1	0.00	11	12	1
40	10	0.00000	288	0.19	5,618	2,893	288
41	1	−5.53443	1	0.00	7	6	1

Boldface indicates instance that was not solved successfully

Table 2 Experimental results for *Random Linkage* with $\sigma = 1$ with GENCAN

Prob	n	Random linkage ($\sigma = 1$)					
		$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$
1	2	−1.14265	1	0.00	10	10	1
2	2	−377.60873	1	0.00	9	5	1
3	2	−186.73091	19	0.00	184	125	31
4a	2	−186.73091	252	0.09	2,603	1,731	512
4b	2	−186.73091	40	0.01	522	302	125
5	5	0.00000	44	0.04	1,141	866	65
5	8	0.00000	24	0.02	588	415	27
5	10	0.00000	4	0.01	152	106	4
5	20	0.00000	50	0.30	3,533	2,098	50
5	30	0.00000	6	0.16	1,196	701	6
6	4	−21.50236	924	1.48	37,607	23,781	6,915
7	2	−24.06250	37	0.01	323	235	74
8	7	−274.16316	3	0.00	36	29	3
9	2	−176.54179	1	0.00	9	7	1
10	2	−3.30687	514	0.21	5,783	3,904	1,426
11	3	−0.49026	2	0.00	23	17	2
12	**2**	**0.00986**	**207**	**0.07**	**2,716**	**1,435**	**423**
13	**2**	**259.52386**	**255**	**60.59**	**1,000,645**	**1,000,563**	**266**
14	4	0.00031	419	0.58	17,945	9,893	1,859
15	4	85,822.20160	1	0.00	12	12	1
16	5	0.00000	18	0.00	227	153	26
17	2	0.00000	1	0.00	7	7	1
18	2	0.00000	1	0.00	6	6	1
19a	4	−10.15320	3	0.00	43	26	3

Table 2 continued

Prob	n	Random linkage ($\sigma = 1$)					
		$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$
19b	4	−10.53641	3	0.00	54	29	3
20	4	0.00000	1	0.00	15	15	1
21	2	−1.03163	2	0.00	21	16	2
22	2	−186.73091	19	0.00	184	125	31
23	2	−78.33233	3	0.00	36	31	3
23	3	−117.49850	3	0.00	38	33	3
23	4	−156.66466	32	0.00	352	327	40
24	2	0.00000	1	0.00	7	6	1
25	2	−0.40746	1	0.00	14	14	1
26	2	−18.05870	1	0.00	10	10	1
27	2	−227.76575	1	0.00	15	15	1
28	2	−2,429.41477	1	0.00	12	12	1
29	2	−2.00000	47	0.01	560	335	133
30	2	0.39789	1	0.00	12	8	1
31	1	−3.37290	5	0.00	43	30	5
32	2	1.00000	14	0.01	264	223	18
33	1	7.00000	1	0.00	4	3	1
34	4	0.00000	1	0.00	31	21	1
35	2	0.00000	2	0.00	26	21	2
36	4	0.00000	1	0.00	42	27	1
37	10	0.00000	1	0.00	4	4	1
37	20	0.00000	1	0.00	8	8	1
37	30	0.00000	996	1.44	5,729	5,716	1,009
37	40	0.00000	148	0.26	799	799	148
38	10	0.00000	1	0.00	3	3	1
39	10	2.50000	1	0.00	12	12	1
40	10	0.00000	275	0.18	5,676	2,725	330
41	1	−5.53443	1	0.00	8	6	1

Boldface indicates instance that was not solved successfully

Lissajous curves, and one used Continuous GRASP. The random linkage heuristic with $\sigma = 0$ is simply a random multistart heuristic. In these tables, the first column indicates the test problem, numbered according to the list of functions in the appendix, n is the number of variables, $f(x^*)$ is the value of the best local minimum found, it_{x^*} is the number of local minimizations performed before the solution x^* is found, t_{x^*} is the CPU time in seconds until x^* is found, f_{eval} is the total number of functional evaluations until x^* is found, g_{eval} is the total number of gradient evaluations until x^* is found, and $\#x_0$ is the number of initial solutions generated by the global component until x^* is found.

To make an overall comparison of the methods, we resort to *performance profiles* [11]. We generate profiles for all of the six algorithms being compared. To generate profiles for each algorithm, we consider a set of 52 test problems and apply each algorithm to each of the test problems. We say an algorithm solves the problem, or is successful, if it can find a

Table 3 Experimental results for *Random Linkage* with $\sigma = 2$ with GENCAN

Prob	n	Random linkage ($\sigma = 2$)					
		$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$
1	2	−1.14265	1	0.00	10	10	1
2	2	−377.60873	1	0.00	9	5	1
3	2	−186.73091	17	0.00	164	109	31
4a	2	−186.73091	227	0.09	3,040	1,481	1,337
4b	2	−186.73091	29	0.01	414	221	125
5	5	0.00000	53	0.05	1,297	910	112
5	8	0.00000	33	0.03	840	563	58
5	10	0.00000	4	0.01	152	106	4
5	20	0.00000	49	0.29	3,372	2,010	50
5	30	0.00000	5	0.12	932	554	6
6	4	−21.50236	924	1.48	37,607	23,781	6,915
7	2	−24.06250	30	0.01	270	189	74
8	7	−274.16316	3	0.00	36	29	3
9	2	−176.54179	1	0.00	9	7	1
10	2	−3.30687	332	0.12	3,965	2,322	1,426
11	3	−0.49026	2	0.00	23	17	2
12	**2**	**0.00986**	**1,835**	**7.08**	**43,696**	**11,666**	**27,922**
13	**2**	**259.52386**	**245**	**61.53**	**1,000,635**	**1,000,543**	**266**
14	4	0.00031	788	2.23	4,8628	18,909	17,819
15	4	85,822.20160	1	0.00	12	12	1
16	5	0.00000	14	0.00	187	124	26
17	2	0.00000	1	0.00	7	7	1
18	2	0.00000	1	0.00	6	6	1
19a	4	−10.15320	3	0.00	43	26	3
19b	4	−10.53641	3	0.00	54	29	3
20	4	0.00000	1	0.00	15	15	1
21	2	−1.03163	2	0.00	16	13	3
22	2	−186.73091	17	0.00	164	109	31
23	2	−78.33233	3	0.00	36	31	3
23	3	−117.49850	3	0.00	38	33	3
23	4	−156.66466	35	0.02	410	361	72
24	2	0.00000	1	0.00	7	6	1
25	2	−0.40746	1	0.00	14	14	1
26	2	−18.05870	1	0.00	10	10	1
27	2	−227.76575	1	0.00	15	15	1
28	2	−2429.41477	1	0.00	12	12	1
29	2	−2.00000	31	0.01	436	234	133
30	2	0.39789	1	0.00	12	8	1
31	1	−3.37290	3	0.00	31	21	5
32	2	1.00000	12	0.01	245	206	18
33	1	7.00000	1	0.00	4	3	1

Table 3 continued

Prob	n	Random linkage ($\sigma = 2$)					
		$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$
34	4	0.00000	1	0.00	31	21	1
35	2	0.00000	2	0.00	26	21	2
36	4	0.00000	1	0.00	42	27	1
37	10	0.00000	1	0.00	4	4	1
37	20	0.00000	1	0.00	8	8	1
37	30	0.00000	988	1.43	5,695	5,674	1,009
37	40	0.00000	148	0.26	799	799	148
38	10	0.00000	1	0.00	3	3	1
39	10	2.50000	1	0.00	12	12	1
40	10	0.00000	233	0.16	4,929	2314	330
41	1	−5.53443	1	0.00	8	6	1

Boldface indicates instance that was not solved successfully

Table 4 Experimental results for *Random Linkage* with $\sigma = 4$ with GENCAN

Prob	n	Random linkage ($\sigma = 4$)					
		$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$
1	2	−1.14265	1	0.00	10	10	1
2	2	−377.60873	1	0.00	9	5	1
3	2	−186.73091	11	0.00	119	69	31
4a	2	−186.73091	101	0.04	2,143	521	1,463
4b	2	−186.73091	23	0.01	364	178	125
5	5	0.00000	36	0.04	964	638	152
5	8	0.00000	31	0.04	785	527	58
5	10	0.00000	4	0.01	152	106	4
5	20	0.00000	48	0.28	3,232	1,922	50
5	30	0.00000	5	0.13	932	554	6
6	4	−21.50236	566	1.44	33,301	14,644	14,299
7	2	−24.06250	23	0.01	216	140	74
8	7	−274.16316	3	0.00	36	29	3
9	2	−176.54179	1	0.00	9	7	1
10	2	−3.30687	1,504	1.26	27,177	9,794	17,322
11	3	−0.49026	2	0.00	23	17	2
12	**2**	**0.00986**	**51**	**0.02**	**1,946**	**354**	**1,389**
13	**2**	**259.52386**	**230**	**66.27**	**1,000,620**	**1,000,513**	**266**
14	4	0.00031	578	1.71	40,787	13,907	17,819
15	4	85,822.20160	1	0.00	12	12	1
16	5	0.00000	13	0.00	175	114	26
17	2	0.00000	1	0.00	7	7	1
18	2	0.00000	1	0.00	6	6	1

Table 4 continued

Prob	n	Random linkage ($\sigma = 4$)					
		$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$
19a	4	−10.15320	3	0.00	43	26	3
19b	4	−10.53641	3	0.00	54	29	3
20	4	0.00000	1	0.00	15	15	1
21	2	−1.03163	2	0.00	16	13	3
22	2	−186.73091	11	0.00	119	69	31
23	2	−78.33233	3	0.00	36	31	3
23	3	−117.49850	3	0.00	38	33	3
23	4	−156.66466	57	0.04	2,047	590	1,485
24	2	0.00000	1	0.00	7	6	1
25	2	−0.40746	1	0.00	14	14	1
26	2	−18.05870	1	0.00	10	10	1
27	2	−227.76575	1	0.00	15	15	1
28	2	−2,429.41477	1	0.00	12	12	1
29	2	−2.00000	20	0.01	337	156	133
30	2	0.39789	1	0.00	12	8	1
31	1	−3.37290	2	0.00	20	14	5
32	2	1.00000	27	0.02	717	363	307
33	1	7.00000	1	0.00	4	3	1
34	4	0.00000	1	0.00	31	21	1
35	2	0.00000	2	0.00	26	21	2
36	4	0.00000	1	0.00	42	27	1
37	10	0.00000	1	0.00	4	4	1
37	20	0.00000	1	0.00	8	8	1
37	30	0.00000	975	1.46	5,634	5,600	1,009
37	40	0.00000	148	0.26	799	799	148
38	10	0.00000	1	0.00	3	3	1
39	10	2.50000	1	0.00	12	12	1
40	**10**	**2.01332**	**848**	**0.66**	**17,021**	**8,177**	**2,262**
41	1	−5.53443	1	0.00	8	6	1

Boldface indicates instance that was not solved successfully

global optimum solution using at most 20,000 calls to GENCAN. For each successful run of an algorithm, we record a given performance measure, such as CPU time or number of calls to GENCAN.

The interpretation of a performance profile graph is as follows. Let the performance measure be distributed in the interval $[0, \bar{p}]$. A point (p, q) in the graph, where $p \in [0, \bar{p}]$ and $q \in [0, 1]$, indicates that the algorithm solved $100 \cdot q\%$ of the problems in at most p units of the performance measure. The most significant regions of the graph are the leftmost and rightmost sides. In the left, we measure the efficiency of the method, i.e. the portion of problems for which each method is the most efficient (for example, the fastest or the one with

Table 5 Experimental results for *Tunneling* with GENCAN

Prob	n	Tunneling with Lissajous curves					
		$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$
1	2	−1.14265	1	0.00	9	10	1
2	2	−377.60873	1	0.00	8	5	1
3	2	−186.73091	2	0.00	112	13	1
4a	2	−186.73091	25	0.02	14,307	164	7
4b	2	−186.73091	7	0.01	3,590	42	2
5	5	0.00000	3	0.00	2,556	44	2
5	8	0.00000	33	0.23	55,184	578	27
5	10	0.00000	4	0.03	6,148	106	4
5	20	0.00000	50	1.06	1,01,483	2,098	50
5	30	0.00000	6	0.28	11,190	701	6
6	4	−21.50236	78	0.17	72,128	1,099	40,055
7	2	−24.06250	5	0.00	1,029	25	1
8	7	−274.16316	3	0.00	4,033	29	3
9	2	−176.54179	1	0.00	8	7	1
10	2	−3.30687	43	0.06	35,999	281	15
11	3	−0.49026	2	0.00	102	12	1
12	**2**	**0.00740**	**581**	**0.45**	**341,896**	**3,530**	**122**
13	2	124.36218	34	12.98	216,037	189,791	14
14	4	0.00031	1,269	33.24	2,987,308	484,777	1,223
15	4	85,822.20160	1	0.00	11	12	1
16	5	0.00000	32	0.10	51,421	249	26
17	2	0.00000	1	0.00	6	7	1
18	2	0.00000	1	0.00	5	6	1
19a	4	−10.15320	3	0.00	4,040	26	3
19b	4	−10.53641	3	0.00	4,051	29	3
20	4	0.00000	1	0.00	14	15	1
21	2	−1.03163	2	0.00	100	12	1
22	2	−186.73091	2	0.00	112	13	1
23	2	−78.33233	3	0.00	4,033	31	3
23	3	−117.49850	3	0.00	4,035	33	3
23	4	−156.66466	40	0.12	78,404	417	40
24	2	0.00000	1	0.00	6	6	1
25	2	−0.40746	1	0.00	13	14	1
26	2	−18.05870	1	0.00	9	10	1
27	2	−227.76575	1	0.00	14	15	1
28	2	−2429.41477	1	0.00	11	12	1
29	2	−2.00000	403	0.34	334,650	2,658	133
30	2	0.39789	1	0.00	11	8	1
31	1	−3.37290	2	0.00	11	11	1
32	2	1.00000	31	0.06	3830	331	15
33	1	7.00000	1	0.00	3	3	1

Table 5 continued

Prob	n	Tunneling with Lissajous curves					
		$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$
34	4	0.00000	1	0.00	30	21	1
35	2	0.00000	2	0.00	11	11	1
36	4	0.00000	1	0.00	41	27	1
37	10	0.00000	1	0.00	3	4	1
37	20	0.00000	1	0.00	7	8	1
37	30	0.00000	1,009	18.02	2,020,778	5,787	1,009
37	40	0.00000	148	3.68	294,651	799	148
38	10	0.00000	1	0.00	2	3	1
39	10	2.50000	1	0.00	11	12	1
40	10	0.00000	562	2.04	556,838	5,627	255
41	1	−5.53443	1	0.00	7	6	1

Boldface indicates instance that was not solved successfully

Table 6 Experimental results for *CGRASP–GENCAN*

Prob	n	CGRASP–GENCAN					
		$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$
1	2	−1.14265	1	0.00	72	7	1
2	2	−377.60873	1	0.00	26	7	1
3	2	−186.73091	1	0.00	72	6	1
4a	2	−186.73091	10	0.00	1,324	38	2
4b	2	−186.73091	6	0.00	655	26	1
5	5	0.00000	1	0.00	305	27	1
5	8	0.00000	1	0.00	596	27	1
5	10	0.00000	1	0.00	666	21	1
5	20	0.00000	1	0.02	2,885	45	1
5	30	0.00000	1	0.07	6,846	50	1
6	4	−21.50236	5	0.00	1,193	16	1
7	2	−24.06210	5	0.00	381	17	1
8	7	−274.16316	1	0.00	211	11	1
9	2	−176.54179	1	0.00	71	6	1
10	2	−3.30687	464	0.11	11,092	1,624	74
11	3	−0.49026	6	0.00	116	25	2
12	2	0.00000	11	0.03	63,057	33	2
13	2	124.36218	1	0.01	3,017	8	1
14	**4**	**0.00094**	**338**	**10.28**	**6,293,840**	**1,552**	**39**
15	4	85,822.20160	1	0.02	1,827	10	1
16	5	0.00000	27	0.01	9,686	72	6
17	2	0.00000	1	0.00	46	5	1
18	2	0.00000	1	0.00	22	5	1

Table 6 continued

Prob	n	CGRASP–GENCAN					
		$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$
19a	4	−10.15320	1	0.00	132	10	1
19b	4	−10.53641	1	0.00	126	8	1
20	4	0.00000	1	0.00	85	14	1
21	2	−1.03163	1	0.00	23	6	1
22	2	−186.73091	1	0.00	72	6	1
23	2	−78.33233	2	0.00	264	14	1
23	3	−117.49810	1	0.00	261	6	1
23	4	−156.66466	2	0.00	685	12	1
24	2	0.00000	1	0.00	38	3	1
25	2	−0.40746	1	0.00	45	3	1
26	2	−18.05870	1	0.00	40	5	1
27	2	−227.76575	1	0.00	137	12	1
28	2	−2,429.41477	1	0.00	133	8	1
29	2	−2.00000	2	0.00	80	8	1
30	2	0.39789	1	0.00	129	4	1
31	1	−3.37290	1	0.00	45	4	1
32	2	1.00000	11	0.00	482	29	3
33	1	7.00000	1	0.00	9	2	1
34	4	0.00000	1	0.00	90	13	1
35	2	0.00000	1	0.00	31	10	1
36	4	0.00000	1	0.00	88	23	1
37	10	0.00000	1	0.00	1,395	4	1
37	20	0.00000	3	0.05	13,579	10	1
37	30	0.00000	2	0.13	21,407	7	1
37	40	0.00000	2	0.28	35,356	7	1
38	10	0.00000	1	0.00	781	3	1
39	10	2.50000	1	0.00	770	11	1
40	10	0.00000	3	0.00	3,136	18	1
41	1	−5.53443	1	0.00	10	4	1

Boldface indicates instance that was not solved successfully

fewest calls to GENCAN). On the right, we measure robustness, i.e. the portion of problems that each algorithm is able to solve successfully.

Figure 3 shows the performance profiles for the six algorithms using the number of calls to GENCAN as the performance measure. The plot clearly shows that CGRASP–GENCAN outperforms the other five heuristics both with respect to efficiency and robustness. Of the other five heuristics, only Tunneling with GENCAN matches CGRASP–GENCAN in robustness (however Tunneling is not as efficient as CGRASP).

Figure 4 shows the performance profiles for the six algorithms using CPU times as the performance measure. The plot clearly shows that CGRASP–GENCAN outperforms the other five heuristics both with respect to efficiency and robustness. Of the other five heuristics, only

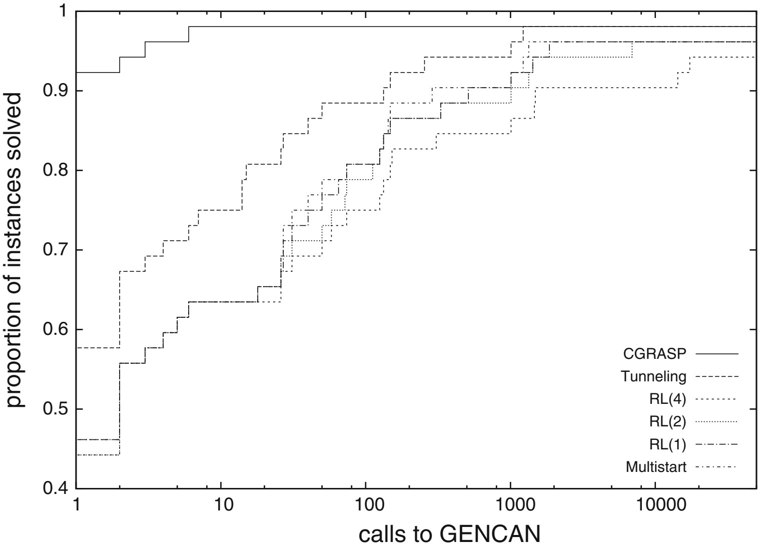

Fig. 3 Performance profiles comparing all the methods using number of local minimizations (calls to the local solver GENCAN) as a performance measurement

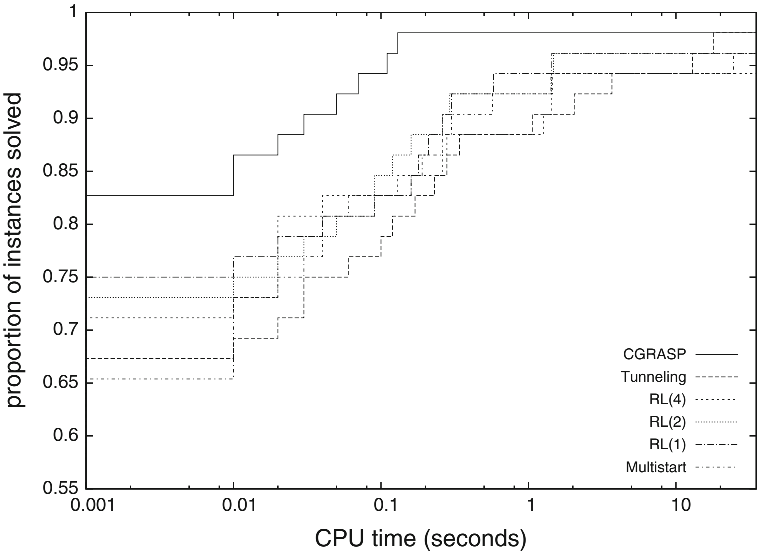

Fig. 4 Performance profiles comparing all the methods using CPU time as a performance measure

Tunneling with GENCAN matches CGRASP–GENCAN in robustness (however Tunneling is not as efficient as CGRASP).

Table 7 summarizes for each of the six algorithms, its efficiency and robustness according to both performance measures.

Table 7 Efficiency and robustness according to performance profiles considering the number of local minimizations and CPU time

	Global method: Local method:	Multistart GENCAN	RL(1) GENCAN	RL(2) GENCAN	RL(4) GENCAN	Tunneling GENCAN	CGRASP GENCAN
Robustness		0.96	0.96	0.96	0.94	0.98	0.98
Efficiency	# Of calls to GENCAN:	0.40	0.40	0.40	0.40	0.54	0.87
w.r.t.	CPU time:	0.58	0.65	0.63	0.63	0.65	0.79

Efficiency and robustness measures vary from 0 to 1. The higher the measure, the better the algorithm

4 Concluding remarks

In this paper, we propose the CGRASP–GENCAN method for continuous bound-constrained global optimization of smooth functions. This method combines the global mechanism of continuous GRASP [15,16] with the local active-set method GENCAN [6], resulting in a method that finds highly accurate solutions that satisfy local first-order optimality conditions. The algorithm was implemented in Fortran and was tested extensively.

Computational testing demonstrates the robustness and efficiency of the proposed method. On 52 test functions used to test previous methods, we compared CGRASP–GENCAN with five other methods, all using GENCAN as their local minimization procedure. Each method was run on the suite of problems under the limitation that at most 20,000 calls to GENCAN be allowed. Performance profiles [11] of the six methods tested clearly show that CGRASP–GENCAN was the most robust of the methods tested, solving 98% of the instances. Furthermore, the profiles show that CGRASP–GENCAN was also the most efficient with respect to number of calls to GENCAN as well as to CPU time.

A: Function definitions

This appendix list all the functions used in the experiments described in this paper. The definition of each function is listed, as well as its domain and global optimum solution value. The functions are numbered from 1 to 41. These numbers are used to identify the functions in Tables 1, 2, 3, 4, 5, and 6.

(1) **Quartic Function**
Definition: $f(x) = \frac{x_1^4}{4} - x_1^2 + \frac{x_1}{10} + \frac{x_2^2}{10}$
Domain: $[-10, 10]^2$
Global Minimum: $f(x^*) = -1.142650$

(2) **Six Hump Function**
Definition: $f(x) = (4 - 2.1x_1^2 + x_1^3)x_1^2 + x_1x_2 + (4x_2^2 + 4)x_2^2$
Domain: $-3 \leq x_1 \leq 3, -2 \leq x_2 \leq 2$
Global Minimum: $f(x^*) = -377.60873$

(3) **Shubert Function** ($n = 2$)
Definition: $f(x) = \prod_{i=1}^{n} \sum_{j=1}^{5} (j \cos((j + 1)x_i + j))$

Domain: $[-10, 10]^2$
Global Minimum: $f(x^*) = -186.73091$

(4) **Penalized Shubert Function** ($n = 2$)
Definition: $f(x) = \prod_{i=1}^{n} \sum_{j=1}^{5} (j \cos((j+1)x_i + j))$
$+\beta((x_1 + 1.42513)^2 + (x_2 + 0.80032)^2)$
Domain: $[-10, 10]^2$
Global Minimum: $f(x^*) = -186.73091$
Note: To perform the tests was considered $\beta = 0.5$ (4a) and $\beta = 1.0$ (4b).

(5) **Piccioni Function**
Definition: $f(x) = 10 \sin^2(\pi x_1) - \sum_{i=1}^{n-1} \left[(x_i - 1)^2(1 + 10\sin^2(\pi x_{i+1}))\right] - (x_n - 1)^2$
Domain: $[-10, 10]^n$
Global Minimum: $f(x^*) = 0.0$
Note: To perform the tests was considered $n \in \{5, 8, 10, 20, 30\}$.

(6) **Levy Function**
Definition: $f(x) = \sin^2(3\pi x_1) + \sum_{i=1}^{n-1} \left[(x_i - 1)^2(1 + \sin^2(3\pi x_{i+1}))\right]$
$+(x_n - 1)(1 + \sin^2(2\pi x_n))$
Domain: $[-10, 10]^n$
Global Minimum: $f(x^*) = -11.50236$

(7) **Schubert Function**
Definition: $f(x) = -\sum_{i=1}^{n} \sum_{j=1}^{5} j \sin((j+1)x_i + j)$
Domain: $[-10, 10]^n$
Global Minimum: $f(x^*) = -24.062499$

(8) **Zhu Function** ($n = 7$)
Definition: $f(x) = \frac{1}{2} \sum_{i=1}^{n} (x_i^4 - 16x_i^2 + 5x_i)$
Domain: $[-5, 2]^7$
Global Minimum: $f(x^*) = -274.16316$

(9) **Hansen Function** ($n = 2$)
Definition: $f(x) = \sum_{i=1}^{5} (i \cos((i-1)x_1 + i) \sum_{j=1}^{5} (j \cos((j+1)x_2 + j)$
Domain: $[-10, 10]^2$
Global Minimum: $f(x^*) = -176.541793$

(10) **Trefethen Function** ($n = 2$)
Definition: $f(x) = e^{\sin(50x_1)} + \sin(60e^{x_2}) + \sin(70\sin(x_1)) + \sin(\sin(80x_2))$
$- \sin(10(x_1 + x_2)) + \frac{(x_1^2 + x_2^2)}{4}$
Domain: $[-1, 1]^2$
Global Minimum: $f(x^*) = -3.306868647475$

(11) **Hartman Function** ($n = 3$)
Definition: $f(x) = -\sum_{i=1}^{4} c_i e^{-\sum_{j=1}^{n} A_{ij}(x_j - P_{ij})^2}$
Domain: $[0, 1]^3$
Global Minimum: $f(x^*) = -0.490260$

Note:

$$A = \begin{pmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix}, \ P = \begin{pmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4378 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.03815 & 0.5743 & 0.8828 \end{pmatrix} \text{ e } c = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \end{pmatrix},$$

(12) **Griewank Function ($n = 2$)**
Definition: $f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
Domain: $[-500, 700]^2$
Global Minimum: $f(x^*) = 0.0$

(13) **Moré Function**
Definition: $f(x) = \sum_{i=1}^{m} [2 + 2i - (\mathrm{e}^{ix_1} + \mathrm{e}^{ix_2})]^2$
Domain: $[-1000, 5]^2$
Global Minimum: $f(x^*) = 124.3621823719$ for $m = 10$

(14) **Moré2 Function ($n = 4$)**
Definition: $f(x) = \sum_{i=1}^{11} \left[y_i - \frac{x_1 u_i (u_i + x_2))}{(u_i(u_i + x_3) + x_4)}\right]^2$
Domain: $[-1000, 1000]^4$
Global Minimum: $f(x^*) = 0.000307486$ for $u_i = \frac{1}{b_i}$,
$b = (0.25, 0.5, 1, 2, 4, 6, 8, 10, 12, 14, 16)^T$ and
$y = (0.1957, 0.1947, 0.1735, 0.1600, 0.0844, 0.0627, 0.0456, 0.0342d0,$
$0.0323, 0.0235, 0.0246)^T$.

(15) **Moré3 Function**
Definition: $f(x) = \sum_{i=1}^{m} \left[(x_1 + t_i x_2 - \mathrm{e}^{t_i})^2 + (x_3 + x_4 \sin(t_i) - \cos(t_i))^2\right]^2$
Domain: $[-100, 100]^4$
Global Minimum: $f(x^*) = 85822.20171974$ for $m = 20$ e $t_i = \frac{i}{5}$

(16) **Hentenryck Function ($n = 5$)**
Definition: $f(x) = \frac{1}{400} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$ Domain: $[-10, 10]^5$
Global Minimum: $f(x^*) = 0.0$

(17) **F2 Function**
Definition: $f(x) = (x_1^2 + x^2 - 11)^2 + (x_1 + x_2^2 - 7)^2$ Domain: $[-6, 6]^2$
Global Minimum: $f(x^*) = 0.0$

(18) **F3 Function**
Definition: $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$
Domain: $[-2, 2]^2$
Global Minimum: $f(x^*) = 0.0$

(19) **F4F5 Function ($n = 4$)**
Definition: $f(x) = -\sum_{i=1}^{m} [(x - a_i)^T (x - a_i) + c_i]^{-1}$
Domain: $[0, 10]^4$
Global Minimum: $f(x^*) = -10.1531957$ for $m = 5$ (19a), $f(x^*) = -10.5362836$ for $m = 10$ (19b).

Note:

i	a_i				c_i
1	4.0	4.0	4.0	4.0	0.1
2	1.0	1.0	1.0	1.0	0.2
3	8.0	8.0	8.0	8.0	0.2
4	6.0	6.0	6.0	6.0	0.4
5	3.0	7.0	3.0	7.0	0.4
6	2.0	9.0	2.0	9.0	0.6
7	5.0	5.0	3.0	3.0	0.3
8	8.0	1.0	8.0	1.0	0.7
9	6.0	2.0	6.0	2.0	0.5
10	7.0	3.6	7.0	3.6	0.5

(20) **F6 Function**
Definition: $f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$
Domain: $[-3, 3]^4$ Global Minimum: $f(x^*) = 0.0$

(21) **F7 Function**
Definition: $f(x) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right) x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$
Domain: $-3 \le x_1 \le 3, -2 \le x_2 \le 2$
Global Minimum: $f(x^*) = -1.031628$

(22) **F8 Function**
Definition: $f(x) = \left[\sum_{i=1}^{5} i \cos((i+1)x_1 + i)\right]\left[\sum_{i=1}^{5} i \cos((i+1)x_2 + i)\right]$
Domain: $[-10, 10]^2$
Global Minimum: $f(x^*) = -186.7309$

(23) **F9aF11 Function**
Definition: $f(x) = \frac{1}{2}\sum_{i=1}^{n}(x_i^4 - 16x_i^2 + 5x_i)$
Domain: $[-20, 20]^n$
Global Minimum: $f(x^*) = -78.332331$, $f(x^*) = -117.4984$ and $f(x^*) = -156.66466$ for $n \in \{2, 3, 4\}$

(24) **F12 Function**
Definition: $f(x) = 0.5x_1^2 + 0.5(1 - \cos(2x_1)) + x_2^2$ Domain: $[-5, 5]^2$
Global Minimum: $f(x^*) = 0.0$

(25) **F13 Function**
Definition: $f(x) = 10x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-1}(x_1^2 + x_2^2)^4$
Domain: $[-5, 5]^2$
Global Minimum: $f(x^*) = -0.407461$

(26) **F14 Function**
Definition: $f(x) = 10^2x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-2}(x_1^2 + x_2^2)^4$
Domain: $[-5, 5]^2$
Global Minimum: $f(x^*) = -18.058697$

(27) **F15 Function**
Definition: $f(x) = 10^3 x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-3}(x_1^2 + x_2^2)^4$
Domain: $[-20, 20]^2$
Global Minimum: $f(x^*) = -227.765747$

(28) **F16 Function**
Definition: $f(x) = 10^4 x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-4}(x_1^2 + x_2^2)^4$
Domain: $[-20, 20]^2$
Global Minimum: $f(x^*) = -2429.414749$

(29) **F17 Function**
Definition: $f(x) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2)$
Domain: $[-5, 5]^2$
Global Minimum: $f(x^*) = -2.0$

(30) **F18 Function**
Definition: $$f(x) = \left(x_2 - \frac{5.1x_1^2}{4\pi^2} + \frac{5x_1}{\pi} - 6\right)^2 \times 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10$$
Domain: $[-20, 20]^2$
Global Minimum: $f(x^*) = 0.397887$

(31) **F19 Function**
Definition: $f(x) = -\left[\sum_{i=1}^{5} \sin((i+1)x_1 + i\right]$
Domain: $[-20, 20]$
Global Minimum: $f(x^*) = -3.372897$

(32) **F20 Function**
Definition: $f(x) = e^{(0.5(x_1^2+x_2^2-25))^2} + \sin(4x_1 - 3x_2)^4 + 0.5(2x_1 + x_2 - 10)^2$
Domain: $[0, 6]^2$
Global Minimum: $f(x^*) = 1.0$

(33) **F21 Function**
Definition: $f(x) = x_1^6 - 15x_1^4 + 27x_1^2 + 250$
Domain: $[-5, 5]$
Global Minimum: $f(x^*) = 7.0$

(34) **F22 Function**
Definition: $$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1)$$
Domain: $[-3, 3]^4$
Global Minimum: $f(x^*) = 0.0$

(35) **F23 Function**
Definition: $f(x) = (1.5 - x_1(1 - x_2))^2 + (2.25 - x_1(1 - x_2^2))^2 + (2.625 - x_1(1 - x_2^3))^2$
Domain: $[0, 5]^2$
Global Minimum: $f(x^*) = 0.0$

(36) **F24 Function**
Definition: $f(x) = \sum_{i=1}^{10} (e^{-0.2i} + 2e^{-0.4i} - x_1 e^{-0.2x_2 i} - x_3 e^{-0.2x_4 i})^2$
Domain: $[-20, 7]^4$
Global Minimum: $f(x^*) = 0.0$

(37) **F25aF28 Function**
Definition: $f(x) = \left[\sum_{i=1}^{n} \frac{x_i^2}{2^{i-1}}\right] + \left[\sum_{i=2}^{n} \frac{x_i x_{i-1}}{2^i}\right]$
Domain: $[-20, 7]^n$
Global Minimum: $f(x^*) = 0.0$ for $n \in \{10, 20, 30, 40\}$.

(38) **F29 Function**
Definition: $f(x) = \sum_{i=1}^{10} x_i^2$
Domain: $[-10, 7]^n$
Global Minimum: $f(x^*) = 0.0$.

(39) **F30 Function**
Definition: $f(x) = \sum_{i=1}^{10} \lfloor x_i^2 + 0.5 \rfloor^2$
Domain: $[-10, 7]^n$
Global Minimum: $f(x^*) = 2.5$.

(40) **F31 Function**
Definition: $f(x) = -20e^{-0.2\sqrt{0.1 \sum_{i=1}^{10} x_i^2}} - e^{0.1 \sum_{i=1}^{10} \cos(2\pi x_i)} + 20 + e$
Domain: $[-10, 20]^n$
Global Minimum: $f(x^*) = 0.0$.

(41) **F32 Function**
Definition: $f(x) = \sin(x_1) + \sin\left(\frac{10x_1}{3}\right) + log_{10}(x_1) - 0.84x_1$
Domain: $[0.1, 6]$
Global Minimum: $f(x^*) = -5.534$.

References

1. Akrotirianakis, I., Floudas, C.: Computational experience with a new class of convex underestimators: box-constrained NLP problems. J. Glob. Optim. **29**, 249–264 (2004)
2. Andreani, R., Martínez, J.M., Salvatierra, M., Yano, F. : Global order-value optimization by means of a multistart harmonic oscillator tunneling strategy. In: Liberti, L, Maculan, N (eds.) Global Optimization: From Theory to Implementation, pp. 379–404. Springer, New York (2006)
3. Andreani, R., Birgin, E.G., Martínez, J.M., Schuverdt, M.L.: On augmented Lagrangian methods with general lower-level constraints. SIAM J. Optim. **18**, 1286–1309 (2007)
4. Andreani, R., Birgin, E.G., Martínez, J.M., Schuverdt, M.L.: Augmented Lagrangian methods under the constant positive linear dependence constraint qualification. Math. Program. **111**, 5–32 (2008)
5. Birgin, E.G., Martínez, J.M.: A box constrained optimization algorithm with negative curvature directions and spectral projected gradients. Computing [Suppl] **15**, 49–60 (2001)
6. Birgin, E.G., Martínez, J.M.: Large-scale active-set box-constrained optimization method with spectral projected gradients. Comput. Optim. Appl. **23**, 101–125 (2002)
7. Birgin, E.G., Martínez, J.M.: Structured minimal-memory inexact quasi-Newton method and secant pre-conditioners for augmented Lagrangian optimization. Comput. Optim. Appl. **39**, 1–16 (2008)
8. Birgin, E.G., Martínez, J.M., Raydan, M.: Nonmonotone spectral projected gradient methods on convex sets. SIAM J. Optim. **10**, 1196–1211 (2000)

9. Birgin, E.G., Martínez, J.M., Raydan, M.: Algorithm 813: SPG—Software for convex-constrained optimization. ACM Trans. Math. Softw. **27**, 340–349 (2001)
10. Birgin, E.G., Floudas, C.A., Martínez, J.M.: Global minimization using an augmented lagrangian method with variable lower-level constraints. Mathematical Programming Published online 20 January 2009 (2009). doi:10.1007/s10107-009-0264-y
11. Dolan, E., Moré, J.: Benchmarking optimization software with performance profiles. Math. Program. **91**, 201–213 (2002)
12. Feo, T., Resende, M.: A probabilistic heuristic for a computationally difficult set covering problem. Oper. Res. Lett. **8**, 67–71 (1989)
13. Feo, T., Resende, M.: Greedy randomized adaptive search procedures. J. Glob. Optim. **6**, 109–133 (1995)
14. Floudas, C.A.: Deterministic Global Optimization: Theory, Methods, and Applications. Kluwer, Dordrecht (2000)
15. Hirsch, M., Pardalos, P., Resende, M.: Speeding up Continuous GRASP. Eur. J. Oper. Res. (2007)
16. Hirsch, M., Meneses, C., Pardalos, P., Resende, M.: Global optimization by continuous GRASP. Optim. Lett. **1**, 201–212 (2007)
17. Horst, R., Pardalos, P., Thoai, N.: Introduction to Global Optimization, Nonconvex Optimization and its Applications, vol. 3. Kluwer, Dordrecht (1995)
18. Krejic, N., Martínez, J.M., Mello, M.P., Pilotta, E.A.: Validation of an augmented Lagrangian algorithm with a Gauss–Newton Hessian approximation using a set of hard-spheres problems. Comput. Optim. Appl. **16**, 247–263 (2000)
19. Levy, A.V., Gomez, S.: The tunneling method applied to global optimization. In: Bogus, P. (ed.) Numerical Optimization, pp. 213–244. SIAM, Philadelphia (1985)
20. Levy, A.V., Montalvo, A.: The tunneling algorithm for the global minimization of functions. SIAM J. Sci. Comput. **6**, 15–29 (1985)
21. Locatelli, M., Schoen, F.: Numerical experience with random linkage algorithms for global optimisation. Tech. Rep. 15-98, Dip. di Sistemi e Informatica, Università di Firenze, Italy (1998)
22. Locatelli, M., Schoen, F.: Random linkage: a family of acceptance/rejection algorithms for global optimization. Math. Prog. **85**, 379–396 (1999)
23. Martínez, J.M.: BOX-QUACAN and the implementation of augmented Lagrangian algorithms for minimization with inequality constraints. Comput. Appl. Math. **19**, 31–56 (2000)