A comparison of some algorithms for bound constrained global optimization

Waltraud Huyer

1 Introduction

In this report we compare two stochastic algorithms for global optimization, PGSL by Raphael & Smith [7] and `global` by Csendes [2] with the deterministic algorithm MCS by Huyer & Neumaier [4] on the test set used by Jones et al. [6], which consists of the seven standard test functions from Dixon & Szegö [3] and two test functions from Yao [8] with the standard box bounds as given in [4]. For simplicity, we refer to this test set as the *Jones test set*.

In [4], a comparison of many methods (not including PGSL or `global`) on this test set showed MCS to be best among the algorithms tested. **Janka** [5] made a significantly more extensive comparison of stochastic optimization programs (not including PGSL or the deterministic MCS), where `global` (called CS in [5]) was the clear winner. The present report complements these findings.

Since all the test functions have known global optima, convergence was defined in terms of the relative error from the globally optimal function value, i.e., an algorithm was successful if a function value with relative error smaller than 1 % was found. In addition, the number of function values was limited to 12000.

2 PGSL by Raphael

The first algorithm in our comparison is the PGSL (probabilistic global search Lausanne) algorithm described in [7] for stochastic global search, which can be found at

`http://imac.epfl.ch/imac/Team/Raphael/PGSL/index.jsp`.

A MEX interface `pgslmex.c` for Matlab was written for the program `pgsl.c` to test the 9 above-mentioned test functions. The Matlab call to the program is the following:

```
[fbest,xbest,ncall] = pgsl(u,v,NFC,NSDC,randomseed,threshold,jones,fac);
```

Here `u` and `v` are the (finite) box bounds of the problem, `NFC` is the maximum number of focusing cycles, `NSDC` is the number of subdomain cycles, `randomseed` is the random seed for the random generator, `threshold` is the objective function value to be reached, `jones` is a parameter to choose among the 9 test functions, and `fac` is a factor to set the precision in PGSL to

```
setup->axes[i]->axisPrecision = fac*(v[i]-u[i]);
```

for $i = 1, \dots, n$. `fbest`, `xbest` and `ncall` are the best function value, best point and number of function calls, respectively.

25 runs for each of the test functions were performed, where the parameter values `NFC` $= 20n$ (n the dimension of the problem), `NSDC` $= 300/n$ (which results in a maximum of 12000 function evaluations), `threshold` $= f_{\text{glob}} + 0.01|f_{\text{glob}}|$ and a loop over `randomseed` from 1 to 25 were taken. The first four columns of Table 1 contain the minimal and maximal number of function values needed to find a global minimizer and the median of these numbers and the number of runs where the threshold was not achieved within 12000 function values.

	PGSL				global				MCS
	min	max	med	fail	min	max	med	fail	
Branin	22	440	224	0	40	76	63	0	31
Six-hump camel	8	142	66	0	27	230	66	0	32
Goldstein–Price	2	1672	292	0	38	1298	91	1	40
Shubert	36	1510	472	0	33	917	233	3	59
Hartman3	18	3702	78	0	38	200	89	0	79
Hartman6	94	4472	206	0	102	590	138	0	74
Shekel5	134	7338	1272	0	62	639	342	0	83
Shekel7	444	9566	1340	0	139	962	437	0	106
Shekel10	468	6184	1254	0	99	1332	452	1	103

Table 1: Comparison of PGSL, `global` and MCS on the 9 test functions of Jones et al.

3 `global` by Csendes

A MEX interface `globalmex.f` for MATLAB was written for the f77 version of `global` from

`ftp://ftp.jate.u-szeged.hu/pub/math/optimization/fortran/`,

a modification of the stochastic optimization algorithm by BOENDER et al. [1] described in detail in [2]. Since the random generator is not implemented in GNU f77, the MEX program was changed to call the MATLAB random generator `rand` instead of `random_number` in the subprogram `urdmn.f`. The quasi-Newton method was used as local search algorithm, i.e., the file `local.f` was linked to the program. The additional parameters `NFE` and `vtr` were added in the subroutine `global` in order to obtain the number of function calls `NFE` as output and to be able to specify a function value `vtr` to reach as an additional stopping criterion. The MATLAB call to the program is given by

```
[x,f,nc,nf] = globalmex(u,v,nsampl,nsel,nsig,m,vtr);
```

where `u` and `v` are the box bounds, `nsampl` is the number of sample points to be drawn in one cycle, `nsel` is the number of best points selected from the sample, `nsig` is the accuracy required in the convergence criterion (the convergence criterion is satisfied if on two successive iterations the variable estimates agree, component by component, to `nsig` digits), `m` serves to choose among the Jones test set and `vtr` is the function value to be reached. The columns of `x` contain the local minimizers found, the vector `f` contains the corresponding function values, `nc` is the number of local minimizers that were found (i.e., the number of columns of `x`) and `nf` is the number of function calls. The Fortran program allows 15 variables to be optimized, which was sufficient for our test set; otherwise some dimension parameters have to be changed and the program has to be recompiled.

The algorithm is quite fast but stops without finding a global minimizer if `nsel` is chosen too small, since the program yields at most `nsel` minimizers. The values `nsampl` $= 100n$ (with the restriction $20 \leq$ `nsample` ≤ 10000), `nsig` $= 6$ are recommended and the suggested value of `nsel` is twice the expected number of local minima (with the restriction $1 \leq$ `nsel` ≤ 20). We did not want to waste too many function values and therefore took only `nsampl` $= 10n$; moreover, we set `nsig` $= 4$. Since the number of local minima is in general unknown, we wanted to have a value of `nsel` depending only on the dimension n of the problem, and `nsel` $= \min(5n, 20)$ turned out to be a good choice. Again the threshold `vtr` is set to

$f_{\text{glob}} + 0.01|f_{\text{glob}}|$. The results for `global` can be found in the columns 5 to 8 of Table 1. Again 25 runs were made, where run j was initialized with the seed j. Here the failures are caused by runs that stopped before finding a global minimizer.

We tried out `nsig` $= 2, \dots, 6$, and the results are summarized in Table 2, where the total number of function calls in all $9 \cdot 25 = 225$ runs (ncall) and the total number of failures for all 225 calls (ncall) are given for the default box bounds as well as the perturbed box bounds described in Section 5. We see that, if `nsig` is taken too small, the desired function value is not reached, but function values are wasted if `nsig` is taken too large.

	unperturbed		perturbed	
nsig	ncall	fail	ncall	fail
2	72911	8	67493	9
3	65017	6	50441	3
4	54819	5	50250	3
5	57220	4	53748	3
6	61757	3	60330	3

Table 2: Summary of `global` results with different values of `nsig`

4 MCS

The last column of Table 1 contains the results (taken from [4]) with MCS and the same stopping criterion. Since MCS is a deterministic algorithms, only one run was made. The standard choice of parameters `smax` $= 5n + 10$, `iinit` $= 1$ (standard initialization list) and the default values for `local`, `gamma` and `hess` were used in addition to the stopping criteria defined in the introduction.

	PGSL				global				MCS			
	min	max	med	fail	min	max	med	fail	min	max	med	fail
Branin	46	2608	254	0	46	73	62	0	27	174	38	0
Six-hump camel	14	204	114	0	29	181	65	0	10	45	29	0
Goldstein–Price	130	1638	356	0	69	253	89	0	31	574	62	0
Shubert	20	1356	394	0	33	1776	218	2	33	1715	230	0
Hartman3	40	274	66	0	42	269	89	0	35	230	131	0
Hartman6	96	12000	4038	3	86	424	120	0	70	12000	78	1
Shekel5	600	10532	3184	0	82	594	332	0	84	3809	335	0
Shekel7	606	12000	1438	1	125	1337	364	0	89	2435	387	0
Shekel10	120	9440	1258	0	130	1485	329	1	99	3780	322	0

Table 3: Comparison on the same problems with perturbed bounds

5 Perturbed box bounds

Instead of the default box bounds $[u, v]$, we consider in a second test perturbed box bounds generated by

$$u_i' = u_i + 0.5\eta(v_i - u_i), \quad v_i' = v_i + 0.5\eta(v_i - u_i), \quad i = 1, \dots, n,$$

where η is a random variable that is uniformly distributed in the interval $[-0.5, 0.5]$. The same set of 25 perturbed box bounds was used for the three algorithm, but only one run (with the random seed set to 0 in the case of PGSL and `global`) was carried through for each pair of bounds. The parameters in the algorithms were chosen as before and the results are presented in Table 3.

6 Conclusions

Although PGSL compared well with genetic algorithms in [7], it is clearly the method that performs least well in our comparison. MCS and `global` perform roughly similar on the Jones test set (with an overall slight advantage to MCS) with regard to number of function values used to find a global minimizer and success rate, which was defined as finding a local minimizer within 12000 function calls.

References

[1] C.G.E. Boender, A.H.G. Rinnoy Kan, G.T. Timmer, and L. Stougie, A stochastic method for global optimization, Mathematical Programming 22 (1982), 125–140.

[2] T. Csendes, Nonlinear parameter estimation by global optimization – efficiency and reliability, Acta Cybernetica 8 (1988), 361-370.

[3] L.C.W. Dixon and G.P. Szegö, The global optimization problem: an introduction, in L.C.W. Dixon and G.P. Szegö (eds.), *Towards Global Optimisation 2*, North-Holland, Amsterdam, 1997, 1–15.

[4] W. Huyer and A. Neumaier, Global optimization by multilevel coordinate search, J. Global Optim. 14 (1999), 331–355.

[5] E. Janka, *Vergleich stochastischer Verfahren zur globalen Optimierung*, Diploma Thesis, University of Vienna, 1999,
`http://www.mat.univie.ac.at/~neum/glopt/janka/gopt_eng.html`.

[6] D.R. Jones, C.D. Perttunen and B.E. Stuckman, Lipschitzian optimization without the Lipschitz constant, J. Optim. Th. Appl. 79 (1993), 157–181.

[7] B. Raphael and I.F.C. Smith, A direct stochastic algorithm for global search, Appl. Math. Comp. 145 (2003), 729–758.

[8] Y. Yao, Dynamic tunneling algorithm for global optimization, IEEE Transactions on System, Man, and Cybernetics 19 (1989), 1222–1230.