shgo: Simplicial homology global optimisation

Stefan C. Endres, Carl Sandrock, Walter W. Focke

Institute of Applied Materials, Department of Chemical Engineering, University of Pretoria, Lynnwood Rd, Hatfield, Pretoria, 0002, South Africa

Abstract

The simplicial homology global optimisation (*shgo*) algorithm is a general purpose global optimisation algorithm based on applications of simplicial integral homology and combinatorial topology. The *shgo* algorithm has proven convergence properties on problems with non-linear objective functions and constraints. The software shows highly competitive performance compared to both open source and commercial software capable of solving derivative free black and grey box optimisation problems.

Keywords: Global optimization, shgo, Computational homology

Mathematics Subject Classification (2010) 90C26 Nonconvex programming, global optimisation

1. Motivation and significance

1.1. Global optimisation of constrained derivative free optimisation problems

A wide range of real-world problems can be formally stated as CDFO (constrained derivative free optimisation) problems. Derivative free problems are usually either black-box or noisy for which deterministic optimisation methods are unsuited to solve. A recent review article [1] cites 36 separate studies with significant applications in the fields of mechanical, aerospace, civil, chemical and biomedical engineering as well as computational chemistry.

In general, the optimisation problems are of the form:

$$
\begin{aligned}
\min_{x} \quad & f(\boldsymbol{x}),\ \boldsymbol{x} \in \mathbb{R}^n \\
\text{s.t.} \quad & g_i(\boldsymbol{x}) \geq 0,\ \forall i = 1, ..., m \\
& h_j(\boldsymbol{x}) = 0,\ \forall j = 1, ..., p
\end{aligned}
$$

where:

- $\boldsymbol{x}$ is a vector of one or more variables.
- $f(x)$ is the objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$.
- $g_i(x)$ are the inequality constraints $\boldsymbol{g} : \mathbb{R}^n \rightarrow \mathbb{R}^m$.
- $h_j(x)$ are the equality constraints $\boldsymbol{h} : \mathbb{R}^n \rightarrow \mathbb{R}^p$.

Many black-box algorithms require lower and upper bounds $x_l \leq x_i \leq x_u$ for each element in $\boldsymbol{x}$ to be specified. While this may greatly increase the speed of convergence, it is not a requirement for *shgo*.

The objective function f usually contains computationally expensive models such as large systems of partial differential equations coupled with non-linear equations. Another example is where f is the result of a simulation using closed source proprietary software.

The simplicial homology global optimisation algorithm is appropriate for solving general purpose black-box optimisation problems to global optimality. Most of the theoretical advantages of *shgo* have been proven for the case where $f(\boldsymbol{x})$ is a Lipschitz smooth function [2]. The algorithm is also proven to converge to the global optimum for the more general case where $f(x)$ is non-continuous, non-convex and non-smooth iff the default sampling method is used [].

1.2. Simplicial homology global optimisation

In order to understand the properties of *shgo* some background theory is required. An important facet of *shgo* is the concept of homology group growth which can be used by an optimisation practitioner as a visual guide of the number of local and global solutions to a problem of arbitrarily high dimensions. In addition a measure of the mutli-modality and the geometric sparsity of solutions of the optimisation problem instance can be deduced.

In brief the algorithm utilises concepts from combinatorial integral homology theory to find sub-domains which are, approximately, locally convex and provides characterisations of the objective function as the algorithm progresses. This is accomplished in several steps. First the construction of a simplicial complex $\mathcal{H}$ built up from the sampling points mapped through f as vertices following the constructions described in [2]. Next a homomorphism is found between $\mathcal{H}$ and $\mathcal{K}$; another simplicial complex which exists on an abstract constructed surface $\mathcal{S}$. The n-dimensional manifold $\mathcal{S}$ is a connected g sum of g tori $S := S_0 \# S_1 \# \cdots \# S_{g-1}$. Figures 1 and 2 demonstrate this construction geometrically in the 2-dimensional case. By using an extension of Brouwer's fixed point theorem [3] adapted to handle non-linear constraints, it is proven that each of the "minimiser points" in Figure 1 corresponds to a sub-domain containing a unique local-minima when the problem is adequately sampled. Through the Invariance Theorem [3] and the Eilenberg-Steenrod Axioms [4, 3] we draw another homomorphism between the surfaces of f and $\mathcal{S}$.

We use the known properties of $\mathcal{S}$ to deduce properties of the unknown function f. The most important corresponding property is the homology groups of $\mathcal{S}$ denoted as $\mathbf{H}_i(\mathcal{S})$. The rank of one of the groups $\mathbf{H}_1(\mathcal{S})$ is proven to correspond to the number of local minima in f. As sampling increases and more local minima are found, so does the rank of $\mathbf{H}_1(\mathcal{S})$ increase. When using uniform sampling, this provides an indication of its multi-modality and the sparsity of the solutions. Furthermore it was proven in [2] that the rank of $\mathbf{H}_1(\mathcal{S})$ cannot increase beyond the true number of local minima in f after adequate sampling. Finally, using the Abelian properties of the homology groups we extend all our previous previously proven properties to hold across non-linear discontinuities as demonstrated geometrically in Figure 3. These properties and their extensions were rigorously proven in [].

2. Software description

2.1. Software Architecture

The module contains only one major class called **SHGO** which can be used to initiate an optimisation instance. The **SHGO** class is initiated with the required inputs of an objective function f and the boundaries placed on the variables $\mathbf{x}$ (which can be specified as infinite in one or both directions for any variable x_i). Optional arguments include the constraint functions $\mathbf{g}$ and $\mathbf{h}$ as well as the two built in sampling methods called 'sobol' and 'simplicial'. A custom sampling method can easily be implemented by inputting a function with the same inputs and outputs as the **SHGO.sobol_points_40** method. The number of sampling points and the number of algorithm iterations can also be optionally specified. Finally any local minimisation routine from the available algorithms in **scipy.optimize.minimize** can be specified.

The **SHGO.construct_complex** method can be used to run the algorithm for the selected number of iterations. The **SHGO.iterate** method can also be used to run a single iteration of *shgo*. The **shgo** function in the base file will (i) initiate an instance of **SHGO**, (ii) run **SHGO.construct_complex** and (iii) do a post-processing check to detect possible routine failures or confirm success before returning the results contained in **SHGO.res**.

SHGO.res contains the main results of the optimization routine at the current iteration as well as other convergence information. **SHGO.res.x** contains the solution corresponding to the global minimum, **SHGO.res.f** is

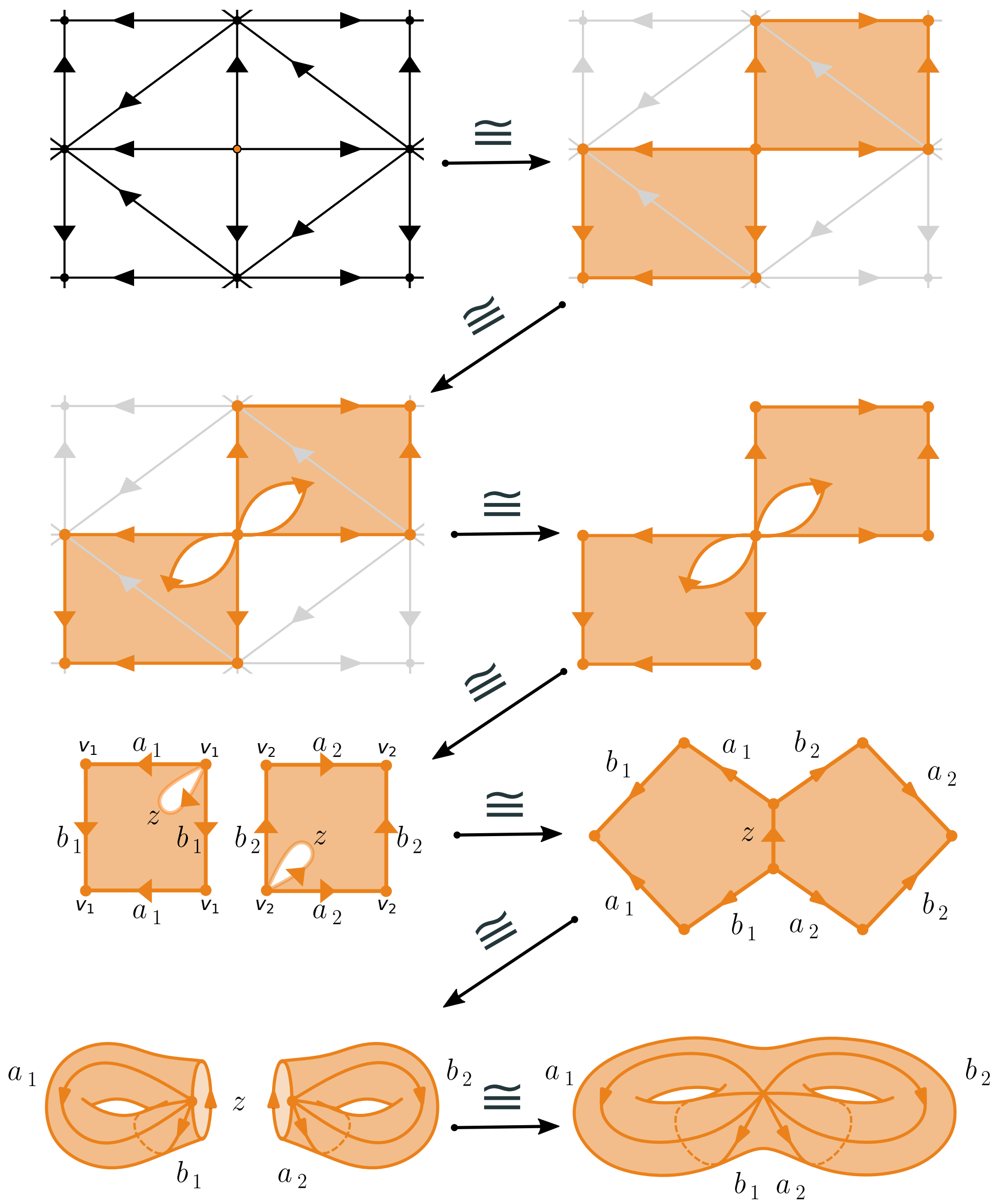

Figure 1: The process of puncturing a hypersphere at a minimiser point in a compact search space. Start by identifying a minimiser point in the $\mathcal{H}^1$ ($\cong$ $\mathcal{K}^1$) graph. By construction, our initial complex exists on the (hyper-)surface of an n-dimensional torus $\mathcal{S}_0$ such that the rest of $\mathcal{K}^1$ is connected and compact. We puncture a hypersphere at the minimiser point and identify the resulting edges (or $(n-1)$-simplices in higher dimensional problems). Next we shrink (*a topological (ie continuous) transformation*) the remainder of the simplicial complex to the faces and vertices of our (hyper-)plane model. Make the appropriate identifications for $\mathcal{S}_0$ and glue the identified and connected face z (a $(n-1)$-simplex) that resulted from the hypersphere puncture. The other faces (ie $(n-1)$-simplices) are connected in the usual way for tori constructions)

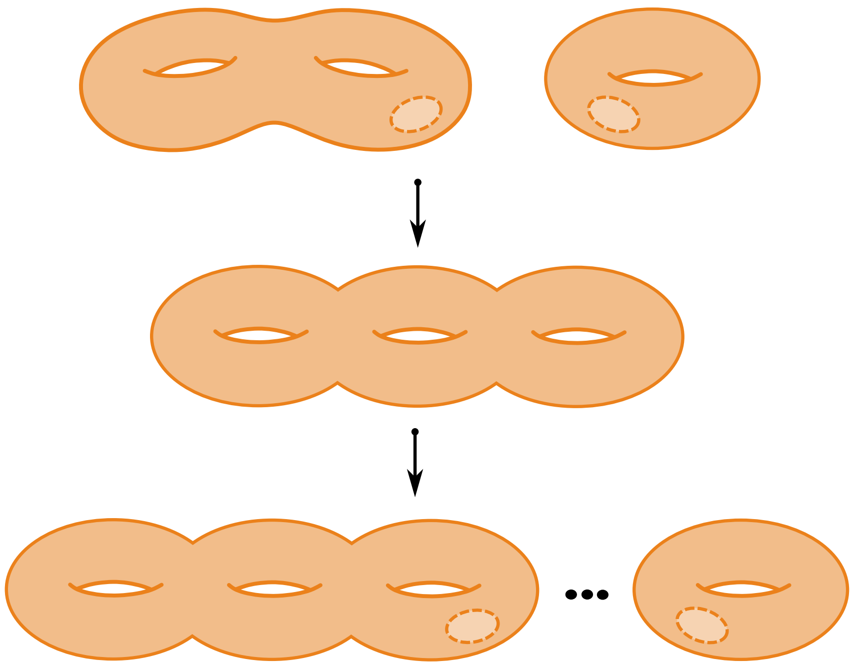

Figure 2: The process of puncturing a new hypersphere on $\mathcal{S}_0 \# \mathcal{S}_1$ can be repeated for any new minimiser point without loss of generality producing $S := S_0 \# S_1 \# \cdots \# S_{g-1} \quad (g \text{ times})$

the function output at the global solution. An ordered list of local minima solutions and their function outputs is also included in **SHGO.res.xl** and **SHGO.res.fl**.

The other classes in the base file of *shgo* are **LMap** and **LMapCache** which contains the data of the local minimisation routines used to map the minimiser starting points to their refined local minima in the main routine.

2.2. Software Functionalities

The *shgo* algorithm is proven to find the globally optimal solution as well as all other local minima in finite processing time. However, an inherit fact of black-box functions is that the true value of the global solution f^* is often unknown. That means that it is unknown how many sampling points and iterations are required to find this solution. The *shgo* module offers several tools in **SHGO** to help optimisation practitioners make intelligent decisions with regards to stopping criteria. In addition, since the properties and stopping criteria of **SHGO** can be adjusted after every iteration, it allows for a versatile algorithm to be used according to the user's needs.

Custom stopping criteria can also be adding by adding a check in **SHGO.stopping_criteria**, which is run after every iteration. The following stopping criteria are built

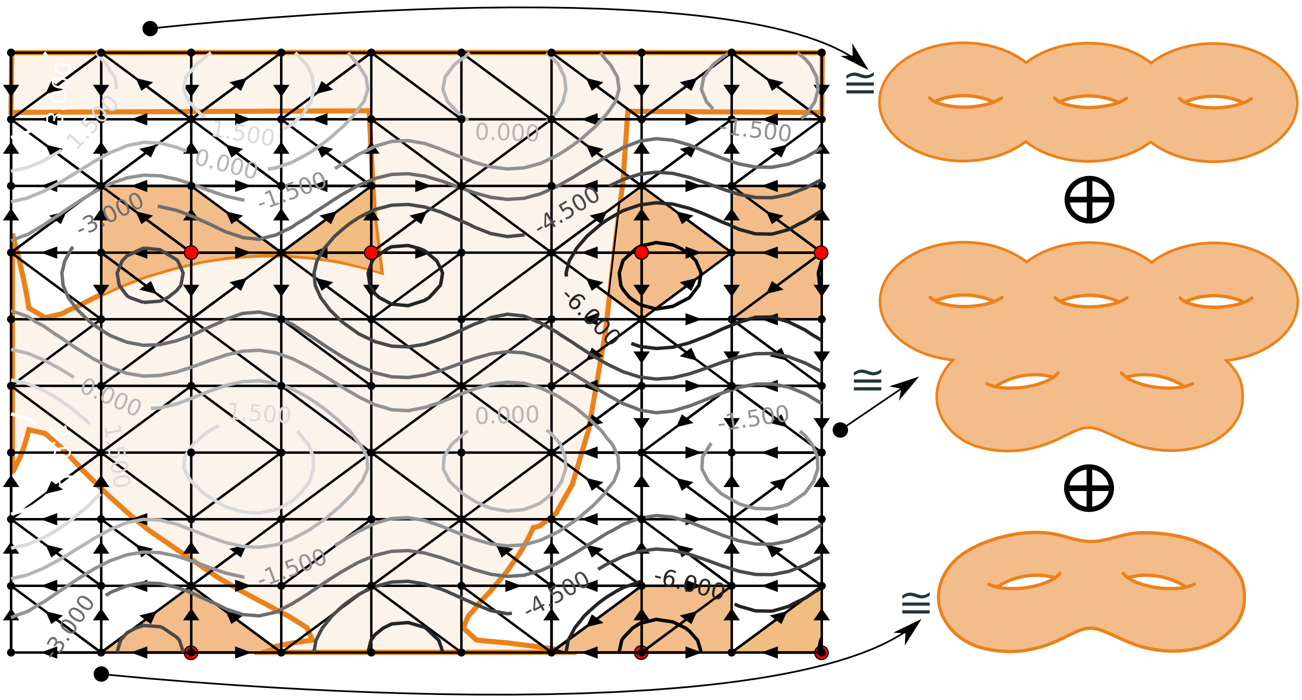

Figure 3: Visual demonstration on surfaces with non-linear constraints, the shaded region is unfeasible. The vertices of the points mapped to infinity have undirected edges, therefore they do not form simplicial complexes in the integral homology. The surfaces of each disconnected simplicial complex $\mathcal{K}_i$ can be constructed from the compact version of the invariance theorem. The rank of the abelian homology groups $\mathbf{H}_1(\mathcal{K}_i)$ is additive over arbitrary direct sums

into **SHGO** and are initiated according to the specified user inputs:

- **SHGO.finite_iterations**
  - Allows for termination after a finite number of iterations.
- **SHGO.finite_fev**
  - Allows for termination after a finite number of objective function evaluations in the feasible domain.
- **SHGO.finite_ev**
  - Allows for termination after a finite number of constraint function evaluations.
- **SHGO.finite_time**
  - Allows for termination after a finite processing runtime has passed.
- **SHGO.finite_precision**
  - If the solution value of the objective function is known (or it is desired to only find a "good enough" solution) and the solution vector(s) $\mathbf{x}^*$ are desired then this criterion will terminate the algorithm within a specified tolerance.
- **SHGO.finite_homology_growth**
  - The homology group rank differential (*hgrd*) which is the global change in rank $(\mathbf{H}_1(\mathcal{S}))$ at every iteration corresponds to the number of new local minima found in every iteration. Therefore it is a measure of the progress in deducing the full geometric information of f. This criterion allows the algorithm to terminate if no new local minima were found after a specified number of iterations. Note that it is inherently impossible to prove that the full geometric structure of a black-box function has been deduced thus this criterion is a heuristic.

Finally the homology group rank (*hgr*) and the homology group rank differential (*hgrd*) can also be tracked in specific volumes of sub-spaces (local change in rank $(\mathbf{H}_1(\mathcal{S}_i \in \mathcal{S})))$. If the 'simplicial' sampling method is used volumes are referred to as cells. A list of cells in each iteration can be accessed at **SHGO.HC.C[i]** where **i** is the iteration number. Every cell contains the **.hg_d** attribute which is the homology group differential in that volume of subspace.

Note that since the low discrepancy sampling is uniform and symmetric after every iteration, the history of the homology group growth can also be used to measure the sparsity of solutions.

3. Illustrative Examples

In order to demonstrate solving problems with non-linear constraints consider the following example from Hock and Schittkowski problem 73 (cattle-feed) [5]:

$$
\begin{aligned}
\text{minimize} : f(x) = & \quad 24.55x_1 + 26.75x_2 + 39x_3 + 40.50x_4 \qquad (1)\\
\text{s.t.} & \quad 2.3x_1 + 5.6x_2 + 11.1x_3 + 1.3x_4 - 5 \geq 0,\\
& \quad 12x_1 + 11.9x_2 + 41.8x_3 + 52.1x_4 - 21\\
& \quad -1.645\sqrt{0.28x_1^2 + 0.19x_2^2 + 20.5x_3^2 + 0.62x_4^2} \geq 0,\\
& \quad x_1 + x_2 + x_3 + x_4 - 1 = 0,\\
& \quad 0 \leq x_i \leq 1 \quad \forall i
\end{aligned}
$$

```
>>> from shgo import shgo
>>> import numpy as np
>>> def f(x):  # (cattle-feed)
...     return 24.55*x[0] + 26.75*x[1] + 39*x[2] + 40.50*x[3]
...
>>> def g1(x):
...     return 2.3*x[0] + 5.6*x[1] + 11.1*x[2] + 1.3*x[3] - 5  #
    >=0
...
>>> def g2(x):
...     return (12*x[0] + 11.9*x[1] +41.8*x[2] + 52.1*x[3] - 21
...             - 1.645 * np.sqrt(0.28*x[0]**2 + 0.19*x[1]**2
...                               + 20.5*x[2]**2 + 0.62*x[3]**2)
...             ) # >=0
...
>>> def h1(x):
...     return x[0] + x[1] + x[2] + x[3] - 1  # == 0
...
>>> cons = ({'type': 'ineq', 'fun': g1},
...         {'type': 'ineq', 'fun': g2},
...         {'type': 'eq', 'fun': h1})
>>> bounds = [(0, 1.0),]*4
>>> res = shgo(f, bounds, iters=3, constraints=cons)
```

```
>>> res
        fun: 29.894378159142136
    funl: array([ 29.89437816])
    message: 'Optimization terminated successfully.'
    nfev: 119
        nit: 3
    nlfev: 40
    nljev: 0
    success: True
        x: array([ 6.35521569e-01,  1.13700270e-13,
   3.12701881e-01,
            5.17765506e-02])
        xl: array([[ 6.35521569e-01,  1.13700270e-13,
   3.12701881e-01,
            5.17765506e-02]])
>>> g1(res.x), g2(res.x), h1(res.x)
(-5.0626169922907138e-14, -2.9594104944408173e-12, 0.0)
```

./example_nlp.py

4. Impact

The potential impact of *shgo* is supported in this section by its performance compared to both commercial and open-source CDFO algorithms.

4.1. Constrained derivative-free optimisation methods for Lipschitz optimsation problems

A recent review and experimental comparison of 22 derivative-free optimisation algorithms by [6] concluded that global optimisation solvers solvers such as TOMLAB/MULTI-MIN, TOMLAB/GLCCLUSTER, MCS and TOMLAB/LGO perform better, on average, than other derivative-free solvers in terms of solution quality within 2500 function evaluations. Both the TOMLAB/GLCCLUSTER and MCS [7] implementations are based on the well-known DIRECT (DIviding RECTangle) algorithm [8].

The DISIMPL (DIviding SIMPLices) algorithm was recently proposed by [9]. The experimental investigation in [9] shows that the proposed simplicial algorithm gives very competitive results compared to the DIRECT algorithm. DISIMPL has been extended in [10, 11]. The Gb-DISIMPL (Globally-biased DISIMPL) was compared in [11] to the DIRECT and DIRECTl methods in extensive numerical experiments on 800 multidimensional multiextremal test functions. Gb-DISIMPL was shown to provide highly competative results compared the other algorithms. More recently the Lc-DISIMPL variant of the algorithm was developed to handle optimisation problems with linear constraints [12]. We used the results from [12] since it contains a CDFO test-suite which compares the most cutting edge open-source as well as the highest performing commercial CDFO algorithms found in literature. Although these problems contain only linear constraints, most of the algorithms in this study can handle non-linear constraints. We used the stopping criteria pe $= 0.01\%$ in this study corresponding to the same tolerance used in [12]. For every test the algorithm was terminated if the global minimum was not found after 100000 objective function evaluations and the test was flagged as a fail again corresponding to the rules in [12].

In Figure 4 we provide experimental results of linearly constrained problems comparing the *shgo*, TGO (topographical global optimization) [13, 14, 15, 16], Lc-DISIMPL [12], LGO (Lipschitz-continuous Global Optimizer) [17], PSwarm [18] (also known as PSO which stands for Partical Swarm Optimization) and DIRECT-L1 [19] algorithms. For the stochastic PSwarm algorithm the average results of 10 runs were used. For DIRECT-L1 we used only the highest performing hyperparameters from the study (pp. $= 10$). It can be seen that *shgo* with the simplicial and Sobol sampling method generally outperforms every other algorithm. The only exception is the better early performance by Lc-DISIMPL. This is attributed to Lc-DISIMPL's initiation step solving the set of equations in the linear constraints. In the test problems where the global minimum lie on a vertex of this convex hull, the algorithm immediately terminates without a global sampling phase. For more general, non-linear constraints it would not be possible to use this feature of Lc-DISIMPL.

4.2. Box constrained derivative-free optimisation methods

In a comparison against other open-source algorithms immediately available in the Python programming language, *shgo* is compared with the TGO, basinhopping (BH) (originally proposed by [20]) and differential evolution (DE) (originally proposed by [21]) global optimisation algorithms. The comparison is done over a large selection of black-box problems from the SciPy [22] global optimisation benchmarking test suite. The problems in this test suite do not contain any constraints (the current SciPy implementations of BH and DE cannot handle non-linear constraints [22]), only bounds that are placed on the variables (known as box problems). We used the stopping criteria pe $= 0.01\%$ in this study. For every test the algorithm was terminated if the global minimum was not found after 10 minutes of processing time and the test was flagged as a fail. Figure 5 and Figure 6 and shows the performance profiles for *shgo*, TGO, DE and BH on the SciPy benchmarking test suite using function evaluations and processing run time as performance criteria. It can be observed that *shgo* and TGO vastly outperform the other

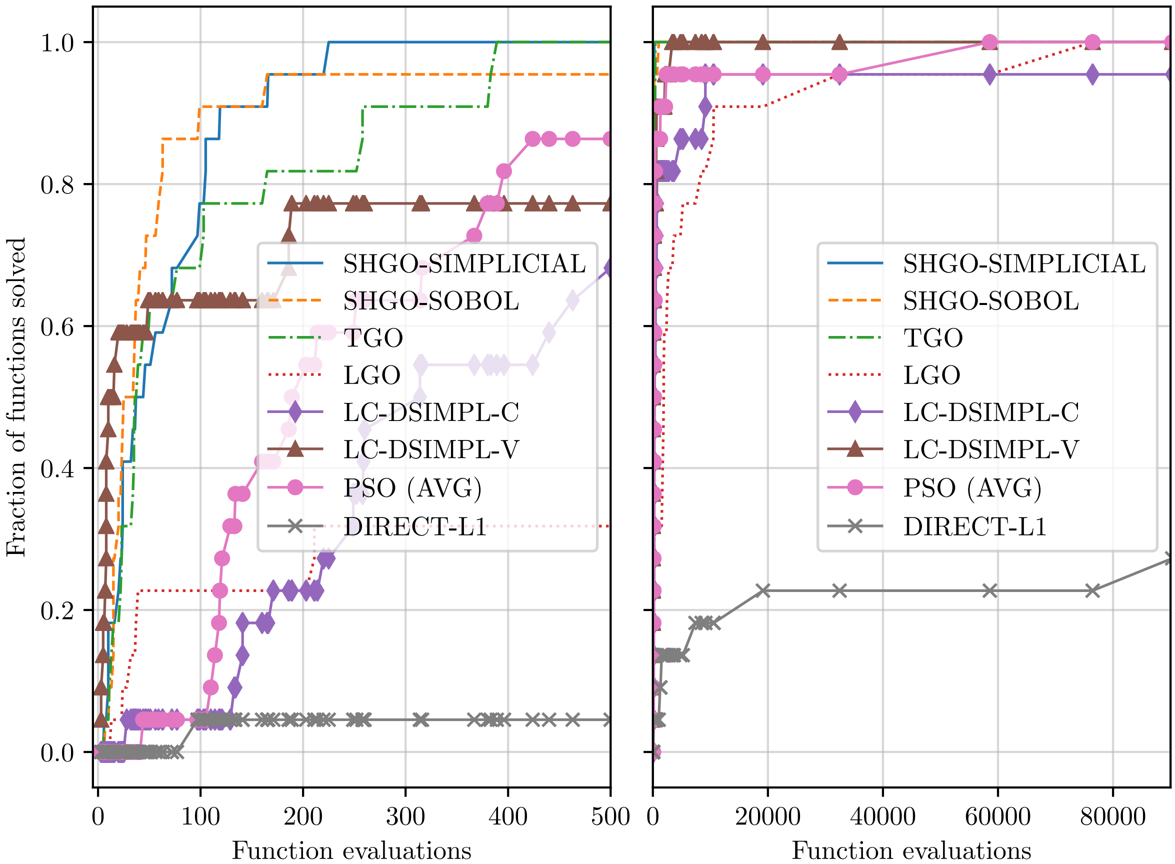

Figure 4: Performance profiles for *shgo*, TGO, Lc-DISIMPL, LGO, PSwarm and DIRECT-L1 algorithms on linearly constrained test problems. The figure displays the fraction test suite problems that can be solved within a given number of objective function evaluations. The results for Lc-DISIMPL-v, PSwarm (avg), DIRECT-L1 were produced by [12]

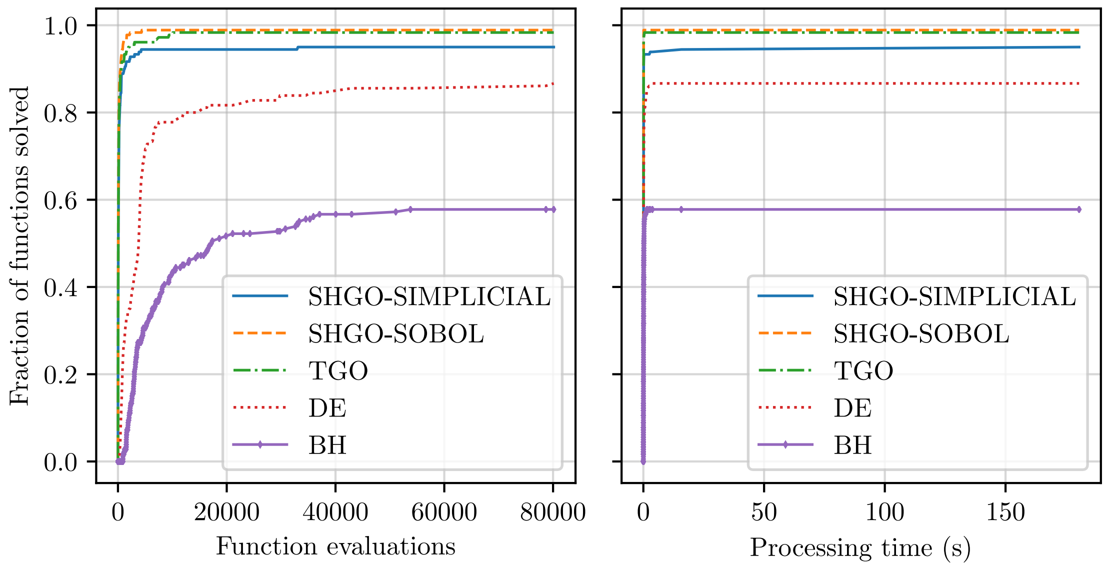

Figure 5: Performance profiles for *shgo*, TGO, DE and BH on SciPy benchmarking test suite

algorithms with *shgo* using the Sobol sampling method having the highest performance throughout.

5. Conclusions

The *shgo* module shows promising properties and performance. It is especially appropriate for computationally expensive black and grey box functions common in science and engineering. The properties and features of *shgo* ca be summarised as follows:

- Convergence to a global minimum is assured for Lipschitz smooth functions.
- Allows for non-linear constraints in the problem statement.
- Extracts all the minima in the limit of an adequately sampled search space (assuming a finite number of local minima).
- Progress can be tracked after every iteration through the calculated homology groups.
- Competitive performance compared to state of the art black-box solvers.

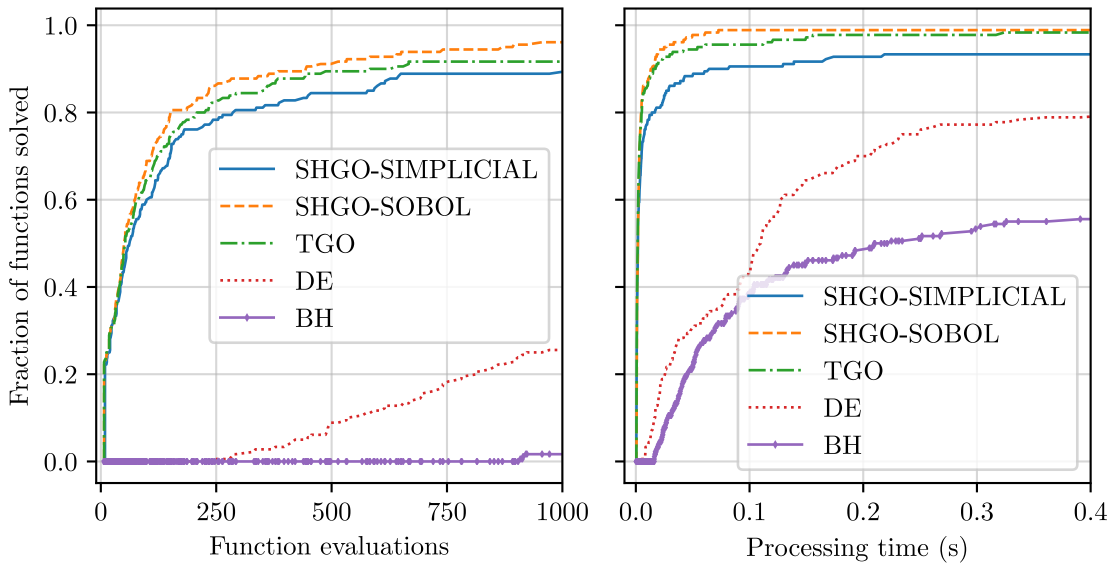

Figure 6: Performance profiles for *shgo*, TGO, DE and BH zoomed in to the range of $f.e. = [0, 1000]$ function evaluations and $[0, 0.4]$ seconds run time

- All of the above properties hold for non-continuous functions with nonlinear constraints assuming the search space contains any sub-spaces that are Lipschitz smooth and convex.

Acknowledgements

Funding:. The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the authors and are not necessarily to be attributed to the NRF. (NRF grant number 92781 Competitive Programme for Rated Researchers (Grant Holder WW Focke)).

References

[1] F. Boukouvala, R. Misener, C. A. Floudas, Global optimization advances in mixed-integer nonlinear programming, minlp, and constrained derivative-free optimization, cdfo, European Journal of Operational Research 252 (3) (2016) 701 – 727. `doi:https://doi.org/10.1016/j.ejor.2015.12.018`. URL `http://www.sciencedirect.com/science/article/pii/S037722171501142X`

[2] S. C. Endres, C. Sandrock, W. W. Focke, A simplicial homology algorithm for lipschitz optimisation, Journal of Global Optimization. URL http://dx.doi.org/10.1007/s10898-018-0645-y

[3] M. Henle, A Combinatorial Introduction to Topology, Unabriged Dover (1994) republication of the edition published by WH Greeman & Company, San Francisco, 1979, 1979.

[4] S. Eilenberg, N. Steenrod, Foundations of algebraic topology, Mathematical Reviews (MathSciNet): MR14: 398b Zentralblatt MATH, Princeton 47.

[5] W. Hock, K. Schittkowski, Test examples for nonlinear programming codes, Journal of Optimization Theory and Applications 30 (1) (1980) 127–129. doi:10.1007/BF00934594. URL https://doi.org/10.1007/BF00934594

[6] L. M. Rios, N. V. Sahinidis, Derivative-free optimization: a review of algorithms and comparison of software implementations, Journal of Global Optimization 56 (3) (2013) 1247–1293.

[7] W. Huyer, A. Neumaier, Global optimization by multilevel coordinate search, Journal of Global Optimization 14 (4) (1999) 331–355. doi:10.1023/A:1008382309369. URL https://doi.org/10.1023/A:1008382309369

[8] D. R. Jones, C. D. Perttunen, B. E. Stuckman, Lipschitzian optimization without the lipschitz constant, Journal of Optimization Theory and Applications 79 (1) (1993) 157–181.

[9] R. Paulavičius, J. Žilinskas, Simplicial lipschitz optimization without the lipschitz constant, Journal of Global Optimization 59 (1) (2014) 23–40.

[10] R. Paulavičius, J. Žilinskas, Simplicial global optimization, Springer, 2014.

[11] R. Paulavičius, Y. D. Sergeyev, D. E. Kvasov, J. Žilinskas, Globally-biased disimpl algorithm for expensive global optimization, Journal of Global Optimization 59 (2) (2014) 545–567.

[12] R. Paulavičius, J. Žilinskas, Advantages of simplicial partitioning for lipschitz optimization problems with linear constraints, Optimization Letters 10 (2) (2016) 237–246.

[13] N. Henderson, M. de Sá Rêgo, W. F. Sacco, R. A. Rodrigues, A new look at the topographical global optimization method and its application to the phase stability analysis of mixtures, Chemical Engineering Science 127 (2015) 151–174.
URL http://linkinghub.elsevier.com/retrieve/pii/S0009250915000494

[14] A. Törn, Clustering methods in global optimization, (in: Preprints of the second ifac symposium on stochastic control, sopron, hungary, part 2), 1986, pp. 138–143.

[15] A. Törn, Topographical global optimization, Reports on Computer Science and Mathematics No 199.

[16] A. Törn, S. Viitanen, Topographical Global Optimization, (in Recent Advances in Global Optimization), Princeton University Press, Princeton, NJ, 1992, pp. 384–398.

[17] J. D. Pintér, Nonlinear optimization with gams /lgo, J. of Global Optimization 38 (1) (2007) 79–101. doi:10.1007/s10898-006-9084-2.
URL http://dx.doi.org/10.1007/s10898-006-9084-2

[18] A. I. Vaz, L. N. Vicente, Pswarm: a hybrid solver for linearly constrained global derivative-free optimization, Optimization Methods and Software 24 (4-5) (2009) 669–685. arXiv:http://dx.doi.org/10.1080/10556780902909948, doi:10.1080/10556780902909948.
URL http://dx.doi.org/10.1080/10556780902909948

[19] D. E. Finkel, Direct optimization algorithm user guide, Center for Research in Scientific Computation, North Carolina State University 2.

[20] D. J. Wales, J. P. Doye, Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms, The Journal of Physical Chemistry A 101 (28) (1997) 5111–5116.

[21] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (4) (1997) 341–359.
URL http://dx.doi.org/10.1023/A:1008202821328

[22] E. Jones, T. Oliphant, P. Peterson, et al., SciPy: Open source scientific tools for Python, [Online; accessed 2016-11-04] (2001–).
URL http://www.scipy.org/

Required Metadata

Current code version

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v0.3.8
C2	Permanent link to code/repository used for this code version	For example: $https://github.com/stefan-endres/shgo$
C3	Legal Code License	MIT
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Python 2.7, 3.5 and 3.6
C6	Compilation requirements, operating environments & dependencies	numpy, scipy, pytest, pytest-cov
C7	If available Link to developer documentation/manual	https://stefan-endres.github.io/shgo/
C8	Support email for questions	stefan.c.endres@gmail.com

Table 1: Code metadata (mandatory)

Current executable software version

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v0.3.8
C2	Permanent link to code/repository used for this code version	https://pypi.python.org/pypi/shgo
C3	Legal Code License	MIT
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Python 2.7, 3.5 and 3.6
C6	Compilation requirements, operating environments & dependencies	numpy, scipy, pytest, pytest-cov
C7	If available Link to developer documentation/manual	https://stefan-endres.github.io/shgo/
C8	Support email for questions	stefan.c.endres@gmail.com

Table 2: Code metadata (mandatory)