CEGO: C++11 Evolutionary Global Optimization

Ian H. Bell[1]

1 National Institute of Standards and Technology, Boulder, CO, USA

DOI: 10.21105/joss.01110

Software

- Review
- Repository
- Archive

Submitted: 29 November 2018
Published: 29 November 2018

Summary

Global optimization is an algorithmic need that is ubiquitous throughout the natural sciences, engineering, and other technical spheres. It is a non-trivial task, particularly when the function to be optimized has many local minima and the optimization algorithm may get trapped in the local minima of the function to be optimized. For that reason, many competing approaches have been proposed for global optimization, especially those inspired by nature. The goal of the library proposed here is to develop a user-friendly framework in C++11 (with wrappers for Python) that can be used to successfully and efficiently carry out global optimization of challenging cost functions with minimum expertise required.

CEGO (C++11 Evolutionary Global Optimization) is a C++11-based optimization library that minimizes an arbitrary cost function. In C++, the cost function to be minimized is of type `std::function<double(const CEGO::AbstractIndividual *)>`, where `CEGO::AbstractIndividual` is the base class for an individual in the population of candidate solutions. The independent variables to be optimized are of type `std::vector<CEGO::numberish>`, where the datatype `CEGO::numberish` can accept both discrete (integer) and continuous values.

A brief summary of the functionality of CEGO includes:

- The implementation of the ALPS algorithm (Hornby, 2006,Hornby (2009a),Hornby (2009b)) for age-layering several optimization runs together. The layers interface is based on migration of younger individuals in the population into older layers. If the individual is too old, and does not dominate another individual in its next layer, it is removed from the population. Age layering can be disabled through the use of a single-layer if desired.
- Latin-hypercube sampling to generate the initial population of individuals in the population.
- A generic architecture for evolving the layered population(s). In the current version, differential evolution (Storn & Price, 1997) is the default evolving method, though an extensible API is available that allows for plug-and-play of alternative population evolution methods. Flags for the evolver are handled in a generic way with a Javascript Object Notation (JSON) structure.
- Use of native C++11 threads (with a thread pool) to parallelize the evaluation of the cost function, allowing for a nearly-linear speedup as more computational cores are made available.
- Ability to log all inputs and outputs (along with an optional filtering function) for further analysis of the progress of the optimization.
- A single-threaded Python wrapper (`PyCEGO`) is written with `pybind11`[1] and is used to demonstrate the functionality of the library, though it cannot fully leverage the parallelism available in CEGO at the C++ level.

[1] https://github.com/pybind/pybind11

A few Jupyter notebooks (Pérez & Granger, 2007) are provided as examples that implement:

A) optimization of cost functions of two- and ten-dimensional continuous variables.
B) the mixed-integer nonlinear optimization problems of the constrained optimization of a pressure vessel mass and dimensionally-constrained spring (Sandgren, 1990)
C) inverse modeling of Gaussian bumps.

All global optimization problems successfully obtain the minimum value from the literature, or better. Furthermore, a binder[2] environment has been configured such that the Jupyter notebooks can be run interactively in an internet browser without any installation on the user's computer.

An example is given here of the global optimization of the modified hundred-digit optimization problem (Townsend, 2014),(Eq. 5.15), a function with 9,318 different local minima in [-1,1] x [-1,1]. CEGO finds the correct global minimum value of -3.398166873463248.

```
from numpy import exp, sin
import PyCEGO

def HundredDigitPlus(c):
    """ The cost function to be minimized """
    x, y = c
    if isinstance(x, PyCEGO.Numberish):
        x = x.as_double()
        y = y.as_double()
    return (0.25*x**2 + exp(sin(100*x)) + sin(140*sin(x))
            + 0.25*y**2 + sin(120*exp(y)) + sin(sin(160*y)) - sin(20*(x+y)))

D = 2
layers = PyCEGO.NumberishLayers(HundredDigitPlus, D, D*20, 1, 3)
layers.set_bounds([PyCEGO.Bound(-1.0, 1.0) for _ in range(D)])
layers.set_builtin_evolver(PyCEGO.BuiltinEvolvers.differential_evolution)

VTR = -4 # Value to reach as acceptable optimization run
for counter in range(1000):
    layers.do_generation()
    cost, coeffs = layers.get_best()
    if counter % 50 == 0:
        print(layers.print_diagnostics())
    if cost < VTR:
        break
```

with the output

```
i: 0 best: -1.81873 c: -0.688891, -0.796059,  queue: 0
i: 50 best: -2.92206 c: -0.588456, 0.030352,  queue: 0
i: 100 best: -3.06838 c: 0.169587, -0.408309,  queue: 0
i: 150 best: -3.3345 c: 0.169926, -0.400595,  queue: 0
i: 200 best: -3.36308 c: -0.145131, -0.404304,  queue: 0
i: 250 best: -3.39762 c: 0.169466, -0.402987,  queue: 0
i: 300 best: -3.39817 c: 0.169674, -0.403046,  queue: 0
```

[2] https://mybinder.org/

```
i: 350 best: -3.39817 c: 0.169674, -0.403046,  queue: 0
i: 400 best: -3.39817 c: 0.169674, -0.403046,  queue: 0
i: 450 best: -3.39817 c: 0.169674, -0.403046,  queue: 0
i: 500 best: -3.39817 c: 0.169674, -0.403046,  queue: 0
i: 550 best: -3.39817 c: 0.169674, -0.403046,  queue: 0
i: 600 best: -3.39817 c: 0.169674, -0.403046,  queue: 0
i: 650 best: -3.39817 c: 0.169674, -0.403046,  queue: 0
i: 700 best: -3.39817 c: 0.169674, -0.403046,  queue: 0
i: 750 best: -3.39817 c: 0.169674, -0.403046,  queue: 0
i: 800 best: -3.39817 c: 0.169674, -0.403046,  queue: 0
i: 850 best: -3.39817 c: 0.169674, -0.403046,  queue: 0
i: 900 best: -3.39817 c: 0.169674, -0.403046,  queue: 0
i: 950 best: -3.39817 c: 0.169674, -0.403046,  queue: 0
```

Disclaimer

References

Hornby, G. S. (2006). ALPS: The Age-Layered Population Structure for Reducing the Problem of Premature Convergence.

Hornby, G. S. (2009a). Steady-State ALPS for Real-Valued Problems. doi:10.1145/1569901.1570011

Hornby, G. S. (2009b). The Age-Layered Population Structure (ALPS) Evolutionary Algorithm.

Pérez, F., & Granger, B. E. (2007). IPython: A system for interactive scientific computing. *Computing in Science and Engineering*, *9*(3), 21–29. doi:10.1109/MCSE.2007.53

Sandgren, E. (1990). Nonlinear integer and discrete programming in mechanical design optimization. *J. Mech. Des.*, *112*(2), 223. doi:10.1115/1.2912596

Storn, R., & Price, K. (1997). Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *J. Global Opt.*, *11*(4), 341–359. doi:10.1023/A:1008202821328

Townsend, A. (2014). *Computing with functions in two dimensions* (PhD thesis). University of Oxford.