pymoo Documentation

Release 0.1.2

Julian Blank

Jun 25, 2018

Contents:

The test problems are uploaded to the PyPi Repository.

```
pip install pymoo
```

1 Implementations

1.1 Algorithms

Genetic Algorithm: A simple genetic algorithm to solve single-objective problems.

NSGA-II *[1]*: Non-dominated sorting genetic algorithm for bi-objective problems. The mating selection is done using the binary tournament by comparing the rank and the crowding distance. The crowding distance is a niching measure in a two-dimensional space which sums up the difference to the neighbours in each dimension. The non-dominated sorting considers the rank determined by being in the ith front and the crowding distance to achieve a good diversity when converging.

NSGA-III *[2] [3]*: A referenced-based algorithm used to solve many-objective problems. The survival selection uses the perpendicular distance to the reference directions. As normalization the boundary intersection method is used [5].

MOEAD/D *[4]*: The classical MOEADD implementation using the Tchebichew decomposition function.

Differential Evolution *[5]*: The classical single-objective differential evolution algorithm where different crossover variations and methods can be defined. It is known for its good results for effective global optimization.

1.2 Methods

Simulated Binary Crossover *[6]*: This crossover simulates a single-point crossover in a binary representation by using an exponential distribution for real values. The polynomial mutation is defined accordingly which performs basically a binary bitflip for real numbers.

2 Usage

```
import time

from matplotlib import animation
import matplotlib.pyplot as plt
import numpy as np


if __name__ == '__main__':

    # load the problem instance
    from pymop.zdt import ZDT1
    problem = ZDT1(n_var=30)

    # create the algorithm instance by specifying the intended parameters
    from pymoo.algorithms.NSGAII import NSGAII
    algorithm = NSGAII("real", pop_size=100, verbose=True)

    start_time = time.time()

    # save the history in an object to observe the convergence over generations
    history = []

    # number of generations to run it
    n_gen = 200

    # solve the problem and return the results
    X, F, G = algorithm.solve(problem,
                              evaluator=(100 * n_gen),
                              seed=2,
                              return_only_feasible=False,
                              return_only_non_dominated=False,
                              history=history)

    print("--- %s seconds ---" % (time.time() - start_time))

    scatter_plot = True
    save_animation = True

    # get the problem dimensionality
    is_2d = problem.n_obj == 2
    is_3d = problem.n_obj == 3
```

```
    if scatter_plot and is_2d:
        plt.scatter(F[:, 0], F[:, 1])
        plt.show()

    if scatter_plot and is_3d:
        fig = plt.figure()
        from mpl_toolkits.mplot3d import Axes3D
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(F[:, 0], F[:, 1], F[:, 2])
        plt.show()

    # create an animation to watch the convergence over time
    if is_2d and save_animation:

        fig = plt.figure()
        ax = plt.gca()

        _F = history[0]['F']
        pf = problem.pareto_front()
        plt.scatter(pf[:,0], pf[:,1], label='Pareto Front', s=60, facecolors='none',␣
↪edgecolors='r')
        scat = plt.scatter(_F[:, 0], _F[:, 1])


        def update(frame_number):
            _F = history[frame_number]['F']
            scat.set_offsets(_F)

            # get the bounds for plotting and add padding
            min = np.min(_F, axis=0) - 0.1
            max = np.max(_F, axis=0) + 0.

            # set the scatter object with padding
            ax.set_xlim(min[0], max[0])
            ax.set_ylim(min[1], max[1])


        # create the animation
        ani = animation.FuncAnimation(fig, update, frames=range(n_gen))

        # write the file
        Writer = animation.writers['ffmpeg']
        writer = Writer(fps=6, bitrate=1800)
        ani.save('%s.mp4' % problem.name(), writer=writer)
```

3 References

4 API

5 Contact

Feel free to contact me if you have any question:

Julian Blank (blankjul [at] egr.msu.edu)
Michigan State University
Computational Optimization and Innovation Laboratory (COIN)
East Lansing, MI 48824, USA

References

[1] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: nsga-ii. *Trans. Evol. Comp*, 6(2):182–197, April 2002. URL: http://dx.doi.org/10.1109/4235.996017, doi:10.1109/4235.996017.

[2] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014. doi:10.1109/TEVC.2013.2281535.

[3] K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, Aug 2014. doi:10.1109/TEVC.2013.2281535.

[4] Qingfu Zhang and Hui Li. A multi-objective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation, Accepted*, 2007.

[5] Kenneth Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. Springer-Verlag, Berlin, Heidelberg, 2005. ISBN 3540209506.

[6] Kalyanmoy Deb, Karthik Sindhya, and Tatsuya Okabe. Self-adaptive simulated binary crossover for real-parameter optimization. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, 1187–1194. New York, NY, USA, 2007. ACM. URL: http://doi.acm.org/10.1145/1276958.1277190, doi:10.1145/1276958.1277190.