Evolutionary computation

Tasks 2024/2025

1 Optimization Problems and Random Search

1.1 Test Optimization Problems

In the first part of this task, you will implement some simple test optimization problems. Prepare a class named Problem, which will store the number of dimensions (d), lower and upper bounds (lowerBound and upperBound), and a name (name). The class should have an abstract method evaluate that calculates the fitness for a given x. Add a method generateRandomSolution that generates a random individual. Implement the following test problems:

- Sphere $L_b = \{-100 \text{ if } i \text{ is even}, -10 \text{ if } i \text{ is odd}\}_{i=1}^{d}$ $U_b = \{100 \text{ if } i \text{ is even}, 10 \text{ if } i \text{ is odd}\}_{i=1}^{n}$
- Ackley $L_b = -32.768$ $U_b = 32.768$ for all dimensions
- Griewank $L_b = -600$ $U_b = 600$ for all dimensions
- Rastrigin $L_b = -5.12$ $U_b = 5.12$ for all dimensions
- Schwefel26 $L_b = -500$ $U_b = 500$ for all dimensions
- Rosenbrock $L_b = -5$ $U_b = 10$ for all dimensions
- Trid $L_b = -d^2$ $U_b = d^2$ for all dimensions
- Bukin $L_b = [-15, -3]$ $U_b = [-5, 3]$
- Carrom Table $L_b = -10$ $U_b = 10$ for all dimensions
- Styblinski-Tang $L_b = -5$ $U_b = 5$ for all dimensions
- Levy $L_b = -10$ $U_b = 10$ for all dimensions
- Michalewicz $L_b = 0$ $U_b = \pi$ for all dimensions

For each problem, use the lower and upper bounds specified here, not those on the linked sources!

List of online resources used to assist with implementing the problems:
Virtual Library of Simulation Experiments: Test Functions and Datasets
Infinity77 Global Optimization Test Functions
Al-Roomi Benchmark Functions for Unconstrained Optimization

To verify the correctness of the problem implementations, you will write Unit tests. For each problem, check whether the fitness at the global optimum is correctly calculated. If a problem allows setting an arbitrary number of dimensions, write tests for 2, 5, and 10 dimensions. For example, for the Sphere problem, if we call the evaluate method with the global optimum position $x^* = (0, \ldots, 0)$, the method should return $f(x^*) = 0$. Small rounding errors in fitness function calculations can affect results when comparing solutions. To ensure accuracy and correctness in comparisons, set a tolerance level directly in the Unit tests or manually calculate the absolute difference. The tolerance should be set to 1e-7.

1.2 Random Search

Implement the random search algorithm (Random Search). Add the classes Solution, Algorithm, and RandomSearch. The Solution class should store the position and fitness of a solution. The Algorithm class should contain a single abstract method execute, which takes a Problem object and the stopping condition maxFes (maximum number of evaluations) and returns a Solution object. The RandomSearch class should inherit from the Algorithm class.

Pseudocode for the Random Search algorithm:

Algorithm 1 Pseudocode for Random Search

```
Input: Problem: containing the fitness function to optimize
Input: maxFes: maximum number of evaluations
Output: bestSolution: the best solution found
Initialize bestSolution ← ∅
for i = 1 to maxFes do
  candidate_i ← generate random solution from Problem
  if Fitness(candidate_i) is better than Fitness(bestSolution) then
    bestSolution ← candidate_i
  end if
end for
return bestSolution
```

Add a parameter isDebug to the algorithm. If the parameter is set to true, the algorithm should output each solution whenever an improvement is found. Example output for a single run of the RandomSearch algorithm:

```
2: x=[50.941405517213894, 42.628581463962036] = 4412.222753698878
9: x=[-12.082394667700441, -52.88616183718793] = 2942.930374775309
11: x=[-42.97669785754075, -9.011739878124942] = 1928.208014369335
18: x=[-0.08894743526496995, 40.96766961582347] = 1678.3578653975055
39: x=[18.18484815981472, -35.531064812651806] = 1593.14526931638
44: x=[5.172293332411201, -10.165800562414432] = 130.09611939129096
188: x=[-1.758732637015143, -8.477419364805172] = 74.95977957527597
251: x=[-2.5646885914133293, -6.620248517067424] = 50.40531799865911
297: x=[-2.2699062648024153, -4.106218121259417] = 22.013501710348468
1358: x=[2.3954048346026724, 0.8447168251345119] = 6.451510836303186
5061: x=[-0.35090602785439273, -0.019513936441057922] = 0.1235158340999735
```

Prepare a main program in which you run the RandomSearch algorithm on all test problems, setting maxFes to **10000**. If a problem allows setting multiple dimensions, run the algorithm with dimensions 2, 5, and 10.

* Deadline: **November 14, 2024**.
* The task is **mandatory** and worth **15 points**.